

# ***MSP430x4xx Family***

## *User's Guide*

## IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

<b>Products</b>		<b>Applications</b>	
Amplifiers	<a href="http://amplifier.ti.com">amplifier.ti.com</a>	Audio	<a href="http://www.ti.com/audio">www.ti.com/audio</a>
Data Converters	<a href="http://dataconverter.ti.com">dataconverter.ti.com</a>	Automotive	<a href="http://www.ti.com/automotive">www.ti.com/automotive</a>
DSP	<a href="http://dsp.ti.com">dsp.ti.com</a>	Broadband	<a href="http://www.ti.com/broadband">www.ti.com/broadband</a>
Interface	<a href="http://interface.ti.com">interface.ti.com</a>	Digital Control	<a href="http://www.ti.com/digitalcontrol">www.ti.com/digitalcontrol</a>
Logic	<a href="http://logic.ti.com">logic.ti.com</a>	Military	<a href="http://www.ti.com/military">www.ti.com/military</a>
Power Mgmt	<a href="http://power.ti.com">power.ti.com</a>	Optical Networking	<a href="http://www.ti.com/opticalnetwork">www.ti.com/opticalnetwork</a>
Microcontrollers	<a href="http://microcontroller.ti.com">microcontroller.ti.com</a>	Security	<a href="http://www.ti.com/security">www.ti.com/security</a>
		Telephony	<a href="http://www.ti.com/telephony">www.ti.com/telephony</a>
		Video & Imaging	<a href="http://www.ti.com/video">www.ti.com/video</a>
		Wireless	<a href="http://www.ti.com/wireless">www.ti.com/wireless</a>

Mailing Address: Texas Instruments  
Post Office Box 655303 Dallas, Texas 75265

# Read This First

---

---

---

---

### ***About This Manual***

This manual discusses modules and peripherals of the MSP430x4xx family of devices. Each discussion presents the module or peripheral in a general sense. Not all features and functions of all modules or peripherals are present on all devices. In addition, modules or peripherals may differ in their exact implementation between device families, or may not be fully implemented on an individual device or device family.

Pin functions, internal signal connections and operational parameters differ from device-to-device. The user should consult the device-specific datasheet for these details.

### ***Related Documentation From Texas Instruments***

For related documentation see the web site <http://www.ti.com/msp430>.

### ***FCC Warning***

This equipment is intended for use in a laboratory test environment only. It generates, uses, and can radiate radio frequency energy and has not been tested for compliance with the limits of computing devices pursuant to subpart J of part 15 of FCC rules, which are designed to provide reasonable protection against radio frequency interference. Operation of this equipment in other environments may cause interference with radio communications, in which case the user at his own expense will be required to take whatever measures may be required to correct this interference.

### ***Notational Conventions***

Program examples, are shown in a special typeface.

**Glossary**

ACLK	Auxiliary Clock	See <i>Basic Clock Module</i>
ADC	Analog-to-Digital Converter	
BOR	Brown-Out Reset	See <i>System Resets, Interrupts, and Operating Modes</i>
BSL	Bootstrap Loader	See <a href="http://www.ti.com/msp430">www.ti.com/msp430</a> for application reports
CPU	Central Processing Unit	See <i>RISC 16-Bit CPU</i>
DAC	Digital-to-Analog Converter	
DCO	Digitally Controlled Oscillator	See <i>FLL+ Module</i>
dst	Destination	See <i>RISC 16-Bit CPU</i>
FLL	Frequency Locked Loop	See <i>FLL+ Module</i>
GIE	General Interrupt Enable	See <i>System Resets Interrupts and Operating Modes</i>
INT(N/2)	Integer portion of N/2	
I/O	Input/Output	See <i>Digital I/O</i>
ISR	Interrupt Service Routine	
LSB	Least-Significant Bit	
LSD	Least-Significant Digit	
LPM	Low-Power Mode	See <i>System Resets Interrupts and Operating Modes</i>
MAB	Memory Address Bus	
MCLK	Master Clock	See <i>FLL+ Module</i>
MDB	Memory Data Bus	
MSB	Most-Significant Bit	
MSD	Most-Significant Digit	
NMI	(Non)-Maskable Interrupt	See <i>System Resets Interrupts and Operating Modes</i>
PC	Program Counter	See <i>RISC 16-Bit CPU</i>
POR	Power-On Reset	See <i>System Resets Interrupts and Operating Modes</i>
PUC	Power-Up Clear	See <i>System Resets Interrupts and Operating Modes</i>
RAM	Random Access Memory	
SCG	System Clock Generator	See <i>System Resets Interrupts and Operating Modes</i>
SFR	Special Function Register	
SMCLK	Sub-System Master Clock	See <i>FLL+ Module</i>
SP	Stack Pointer	See <i>RISC 16-Bit CPU</i>
SR	Status Register	See <i>RISC 16-Bit CPU</i>
src	Source	See <i>RISC 16-Bit CPU</i>
TOS	Top-of-Stack	See <i>RISC 16-Bit CPU</i>
WDT	Watchdog Timer	See <i>Watchdog Timer</i>

---

**Register Bit Conventions**

Each register is shown with a key indicating the accessibility of the each individual bit, and the initial condition:

*Register Bit Accessibility and Initial Condition*

<b>Key</b>	<b>Bit Accessibility</b>
rw	Read/write
r	Read only
r0	Read as 0
r1	Read as 1
w	Write only
w0	Write as 0
w1	Write as 1
(w)	No register bit implemented; writing a 1 results in a pulse. The register bit is always read as 0.
h0	Cleared by hardware
h1	Set by hardware
-0,-1	Condition after PUC
-(0),-(1)	Condition after POR



# Contents

---

---

---

<b>1</b>	<b>Introduction</b>	<b>1-1</b>
1.1	Architecture	1-2
1.2	Flexible Clock System	1-2
1.3	Embedded Emulation	1-3
1.4	Address Space	1-4
1.4.1	Flash/ROM	1-4
1.4.2	RAM	1-4
1.4.3	Peripheral Modules	1-5
1.4.4	Special Function Registers (SFRs)	1-5
1.4.5	Memory Organization	1-5
<b>2</b>	<b>System Resets, Interrupts, and Operating Modes</b>	<b>2-1</b>
2.1	System Reset and Initialization	2-2
2.1.1	Brownout Reset (BOR)	2-3
2.1.2	Device Initial Conditions After System Reset	2-4
2.2	Interrupts	2-5
2.2.1	(Non)-Maskable Interrupts (NMI)	2-6
2.2.2	Maskable Interrupts	2-9
2.2.3	Interrupt Processing	2-10
2.2.4	Interrupt Vectors	2-12
2.2.5	Special Function Registers (SFRs)	2-12
2.3	Operating Modes	2-13
2.3.1	Entering and Exiting Low-Power Modes	2-15
2.4	Principles for Low-Power Applications	2-16
2.5	Connection of Unused Pins	2-16
<b>3</b>	<b>RISC 16-Bit CPU</b>	<b>3-1</b>
3.1	CPU Introduction	3-2
3.2	CPU Registers	3-4
3.2.1	Program Counter (PC)	3-4
3.2.2	Stack Pointer (SP)	3-5
3.2.3	Status Register (SR)	3-6
3.2.4	Constant Generator Registers CG1 and CG2	3-7
3.2.5	General-Purpose Registers R4 - R15	3-8
3.3	Addressing Modes	3-9
3.3.1	Register Mode	3-10
3.3.2	Indexed Mode	3-11
3.3.3	Symbolic Mode	3-12
3.3.4	Absolute Mode	3-13

3.3.5	Indirect Register Mode .....	3-14
3.3.6	Indirect Autoincrement Mode .....	3-15
3.3.7	Immediate Mode .....	3-16
3.4	Instruction Set .....	3-17
3.4.1	Double-Operand (Format I) Instructions .....	3-18
3.4.2	Single-Operand (Format II) Instructions .....	3-19
3.4.3	Jumps .....	3-20
3.4.4	Instruction Cycles and Lengths .....	3-72
3.4.5	Instruction Set Description .....	3-74
<b>4</b>	<b>FLL+ Clock Module .....</b>	<b>4-1</b>
4.1	FLL+ Clock Module Introduction .....	4-2
4.2	FLL+ Clock Module Operation .....	4-5
4.2.1	FLL+ Clock features for Low-Power Applications .....	4-5
4.2.2	LFXT1 Oscillator .....	4-6
4.2.3	XT2 Oscillator .....	4-6
4.2.4	Digitally-Controlled Oscillator (DCO) .....	4-7
4.2.5	Frequency Locked Loop (FLL) .....	4-7
4.2.6	DCO Modulator .....	4-8
4.2.7	Disabling the FLL Hardware and Modulator .....	4-9
4.2.8	FLL Operation from Low-Power Modes- .....	4-9
4.2.9	Buffered Clock Output .....	4-9
4.2.10	FLL+ Fail-Safe Operation .....	4-10
4.3	FLL+ Clock Module Registers .....	4-11
<b>5</b>	<b>Flash Memory Controller .....</b>	<b>5-1</b>
5.1	Flash Memory Introduction .....	5-2
5.2	Flash Memory Segmentation .....	5-3
5.3	Flash Memory Operation .....	5-4
5.3.1	Flash Memory Timing Generator .....	5-4
5.3.2	Erasing Flash Memory .....	5-5
5.3.3	Writing Flash Memory .....	5-8
5.3.4	Flash Memory Access During Write or Erase .....	5-14
5.3.5	Stopping a Write or Erase Cycle .....	5-15
5.3.6	Configuring and Accessing the Flash Memory Controller .....	5-15
5.3.7	Flash Memory Controller Interrupts .....	5-15
5.3.8	Programming Flash Memory Devices .....	5-15
5.4	Flash Memory Registers .....	5-17
<b>6</b>	<b>Supply Voltage Supervisor .....</b>	<b>6-1</b>
6.1	SVS Introduction .....	6-2
6.2	SVS Operation .....	6-4
6.2.1	Configuring the SVS .....	6-4
6.2.2	SVS Comparator Operation .....	6-4
6.2.3	Changing the VLDx Bits .....	6-5
6.2.4	SVS Operating Range .....	6-6
6.3	SVS Registers .....	6-7



---

<b>7</b>	<b>Hardware Multiplier</b>	<b>7-1</b>
7.1	Hardware Multiplier Introduction	7-2
7.2	Hardware Multiplier Operation	7-3
7.2.1	Operand Registers	7-3
7.2.2	Result Registers	7-4
7.2.3	Software Examples	7-5
7.2.4	Indirect Addressing of RESLO	7-6
7.2.5	Using Interrupts	7-6
7.3	Hardware Multiplier Registers	7-7
<b>8</b>	<b>DMA Controller</b>	<b>8-1</b>
8.1	DMA Introduction	8-2
8.2	DMA Operation	8-4
8.2.1	DMA Addressing Modes	8-4
8.2.2	DMA Transfer Modes	8-5
8.2.3	Initiating DMA Transfers	8-12
8.2.4	Stopping DMA Transfers	8-14
8.2.5	DMA Channel Priorities	8-14
8.2.6	DMA Transfer Cycle Time	8-15
8.2.7	Using DMA with System Interrupts	8-16
8.2.8	DMA Controller Interrupts	8-16
8.2.9	Using the I2C Module with the DMA Controller	8-17
8.2.10	Using ADC12 with the DMA Controller	8-17
8.2.11	Using DAC12 With the DMA Controller	8-17
8.3	DMA Registers	8-18
<b>9</b>	<b>Digital I/O</b>	<b>9-1</b>
9.1	Digital I/O Introduction	9-2
9.2	Digital I/O Operation	9-3
9.2.1	Input Register PxIN	9-3
9.2.2	Output Registers PxOUT	9-3
9.2.3	Direction Registers PxDIR	9-3
9.2.4	Function Select Registers PxSEL	9-4
9.2.5	P1 and P2 Interrupts	9-5
9.2.6	Configuring Unused Port Pins	9-6
9.3	Digital I/O Registers	9-7
<b>10</b>	<b>Watchdog Timer, Watchdog Timer+</b>	<b>10-1</b>
10.1	Watchdog Timer Introduction	10-2
10.2	Watchdog Timer Operation	10-4
10.2.1	Watchdog Timer Counter	10-4
10.2.2	Watchdog Mode	10-4
10.2.3	Interval Timer Mode	10-4
10.2.4	Watchdog Timer Interrupts	10-5
10.2.5	WDT+ Enhancements	10-5
10.2.6	Operation in Low-Power Modes	10-6
10.2.7	Software Examples	10-6
10.3	Watchdog Timer Registers	10-7

<b>11 Basic Timer1</b>	<b>11-1</b>
11.1 Basic Timer1 Introduction	11-2
11.2 Basic Timer1 Operation	11-4
11.2.1 Basic Timer1 Counter One	11-4
11.2.2 Basic Timer1 Counter Two	11-4
11.2.3 16-bit Counter Mode	11-4
11.2.4 Basic Timer1 Operation: Signal fLCD	11-5
11.2.5 Basic Timer1 Interrupts	11-5
11.3 Basic Timer1 Registers	11-6
<b>12 Timer_A</b>	<b>12-1</b>
12.1 Timer_A Introduction	12-2
12.2 Timer_A Operation	12-4
12.2.1 16-Bit Timer Counter	12-4
12.2.2 Starting the Timer	12-5
12.2.3 Timer Mode Control	12-5
12.2.4 Capture/Compare Blocks	12-11
12.2.5 Output Unit	12-13
12.2.6 Timer_A Interrupts	12-17
12.3 Timer_A Registers	12-19
<b>13 Timer_B</b>	<b>13-1</b>
13.1 Timer_B Introduction	13-2
13.1.1 Similarities and Differences From Timer_A	13-2
13.2 Timer_B Operation	13-4
13.2.1 16-Bit Timer Counter	13-4
13.2.2 Starting the Timer	13-5
13.2.3 Timer Mode Control	13-5
13.2.4 Capture/Compare Blocks	13-11
13.2.5 Output Unit	13-14
13.2.6 Timer_B Interrupts	13-18
13.3 Timer_B Registers	13-20
<b>14 USART Peripheral Interface, UART Mode</b>	<b>14-1</b>
14.1 USART Introduction: UART Mode	14-2
14.2 USART Operation: UART Mode	14-4
14.2.1 USART Initialization and Reset	14-4
14.2.2 Character Format	14-4
14.2.3 Asynchronous Communication Formats	14-5
14.2.4 USART Receive Enable	14-9
14.2.5 USART Transmit Enable	14-10
14.2.6 UART Baud Rate Generation	14-11
14.2.7 USART Interrupts	14-17
14.3 USART Registers: UART Mode	14-21

<b>15</b>	<b>USART Peripheral Interface, SPI Mode</b>	<b>15-1</b>
15.1	USART Introduction: SPI Mode	15-2
15.2	USART Operation: SPI Mode	15-4
15.2.1	USART Initialization and Reset	15-4
15.2.2	Master Mode	15-5
15.2.3	Slave Mode	15-6
15.2.4	SPI Enable	15-7
15.2.5	Serial Clock Control	15-9
15.2.6	SPI Interrupts	15-11
15.3	USART Registers: SPI Mode	15-13
<b>16</b>	<b>OA</b>	<b>16-1</b>
16.1	OA Introduction	16-2
16.2	OA Operation	16-4
16.2.1	OA Amplifier	16-4
16.2.2	OA Input	16-4
16.2.3	OA Output	16-4
16.2.4	OA Configurations	16-5
16.3	OA Registers	16-11
<b>17</b>	<b>Comparator_A</b>	<b>17-1</b>
17.1	Comparator_A Introduction	17-2
17.2	Comparator_A Operation	17-4
17.2.1	Comparator	17-4
17.2.2	Input Analog Switches	17-4
17.2.3	Output Filter	17-5
17.2.4	Voltage Reference Generator	17-5
17.2.5	Comparator_A, Port Disable Register CAPD	17-6
17.2.6	Comparator_A Interrupts	17-6
17.2.7	Comparator_A Used to Measure Resistive Elements	17-7
17.3	Comparator_A Registers	17-9
<b>18</b>	<b>LCD Controller</b>	<b>18-1</b>
18.1	LCD Controller Introduction	18-2
18.2	LCD Controller Operation	18-4
18.2.1	LCD Memory	18-4
18.2.2	Blinking the LCD	18-4
18.2.3	LCD Timing Generation	18-4
18.2.4	LCD Voltage Generation	18-5
18.2.5	LCD Outputs	18-5
18.2.6	Static Mode	18-6
18.2.7	2-Mux Mode	18-9
18.2.8	3-Mux Mode	18-12
18.2.9	4-Mux Mode	18-15
18.3	LCD Controller Registers	18-18

<b>19</b>	<b>LCD_A Controller</b>	<b>19-1</b>
19.1	LCD_A Controller Introduction	19-2
19.2	LCD_A Controller Operation	19-4
19.2.1	LCD Memory	19-4
19.2.2	Blinking the LCD	19-4
19.2.3	LCD_A Voltage And Bias Generation	19-5
19.2.4	LCD Timing Generation	19-8
19.2.5	LCD Outputs	19-8
19.2.6	Static Mode	19-9
19.2.7	2-Mux Mode	19-12
19.2.8	3-Mux Mode	19-15
19.2.9	4-Mux Mode	19-18
19.3	LCD Controller Registers	19-21
<b>20</b>	<b>ADC12</b>	<b>20-1</b>
20.1	ADC12 Introduction	20-2
20.2	ADC12 Operation	20-4
20.2.1	12-Bit ADC Core	20-4
20.2.2	ADC12 Inputs and Multiplexer	20-5
20.2.3	Voltage Reference Generator	20-6
20.2.4	Auto Power-Down	20-6
20.2.5	Sample and Conversion Timing	20-7
20.2.6	Conversion Memory	20-10
20.2.7	ADC12 Conversion Modes	20-10
20.2.8	Using the Integrated Temperature Sensor	20-16
20.2.9	ADC12 Grounding and Noise Considerations	20-17
20.2.10	ADC12 Interrupts	20-18
20.3	ADC12 Registers	20-20
<b>21</b>	<b>SD16</b>	<b>21-1</b>
21.1	SD16 Introduction	21-2
21.2	SD16 Operation	21-4
21.2.1	ADC Core	21-4
21.2.2	Analog Input Range and PGA	21-4
21.2.3	Voltage Reference Generator	21-4
21.2.4	Auto Power-Down	21-4
21.2.5	Channel Selection	21-5
21.2.6	Digital Filter	21-6
21.2.7	Conversion Memory Registers: SD16MEMx	21-9
21.2.8	Conversion Modes	21-10
21.2.9	Conversion Operation Using Preload	21-13
21.2.10	Using the Integrated Temperature Sensor	21-15
21.2.11	Interrupt Handling	21-16
21.3	SD16 Registers	21-18

<b>22</b>	<b>SD16_A</b>	<b>22-1</b>
22.1	SD16_A Introduction	22-2
22.2	SD16_A Operation	22-4
22.2.1	ADC Core	22-4
22.2.2	Analog Input Range and PGA	22-4
22.2.3	Voltage Reference Generator	22-4
22.2.4	Auto Power-Down	22-4
22.2.5	Channel Selection	22-5
22.2.6	Analog Input Characteristics	22-5
22.2.7	Digital Filter	22-6
22.2.8	Conversion Memory Register: SD16MEM0	22-10
22.2.9	Conversion Modes	22-11
22.2.10	Using the Integrated Temperature Sensor	22-12
22.2.11	Interrupt Handling	22-13
22.3	SD16_A Registers	22-14
<b>23</b>	<b>DAC12</b>	<b>23-1</b>
23.1	DAC12 Introduction	23-2
23.2	DAC12 Operation	23-5
23.2.1	DAC12 Core	23-5
23.2.2	DAC12 Reference	23-6
23.2.3	Updating the DAC12 Voltage Output	23-6
23.2.4	DAC12_xDAT Data Format	23-7
23.2.5	DAC12 Output Amplifier Offset Calibration	23-8
23.2.6	Grouping Multiple DAC12 Modules	23-9
23.2.7	DAC12 Interrupts	23-10
23.3	DAC12 Registers	23-11
<b>24</b>	<b>Scan IF</b>	<b>24-1</b>
24.1	Scan IF Introduction	24-2
24.2	Scan IF Operation	24-4
24.2.1	Scan IF Analog Front End	24-4
24.2.2	Scan IF Timing State Machine	24-14
24.2.3	Scan IF Processing State Machine	24-20
24.2.4	Scan IF Debug Register	24-26
24.2.5	Scan IF Interrupts	24-27
24.2.6	Using the Scan IF with LC Sensors	24-28
24.2.7	Using the Scan IF With Resistive Sensors	24-32
24.2.8	Quadrature Decoding	24-33
24.3	Scan IF Registers	24-35

# Introduction

---

---

---

---

This chapter describes the architecture of the MSP430.

<b>Topic</b>	<b>Page</b>
1.1 Architecture .....	1-2
1.2 Flexible Clock System .....	1-2
1.3 Embedded Emulation .....	1-3
1.4 Address Space .....	1-4

## 1.1 Architecture

The MSP430 incorporates a 16-bit RISC CPU, peripherals, and a flexible clock system that interconnect using a von-Neumann common memory address bus (MAB) and memory data bus (MDB). Partnering a modern CPU with modular memory-mapped analog and digital peripherals, the MSP430 offers solutions for demanding mixed-signal applications.

Key features of the MSP430x4xx family include:

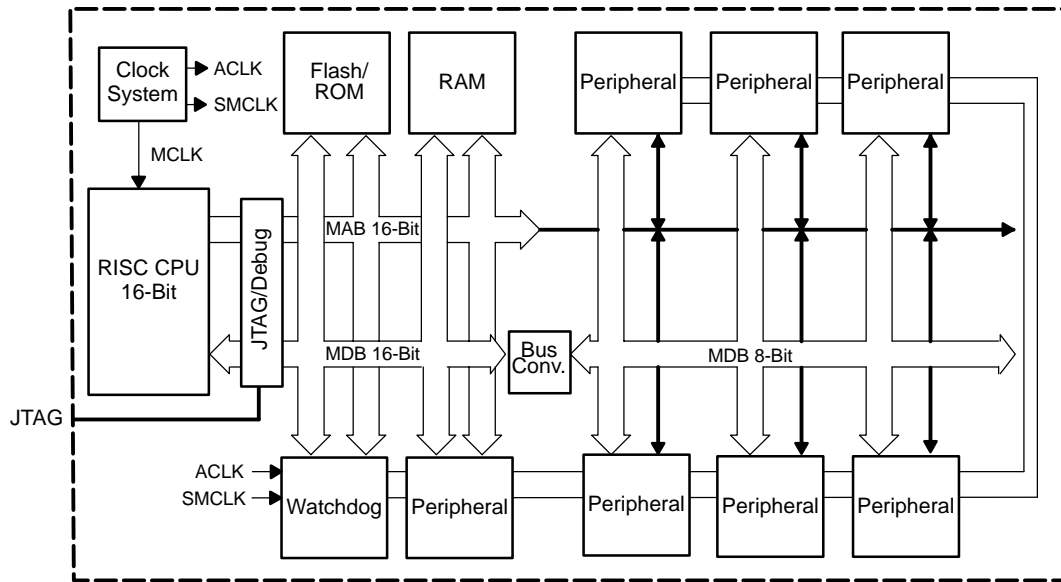
- Ultralow-power architecture extends battery life
  - 0.1- $\mu$ A RAM retention
  - 0.8- $\mu$ A real-time clock mode
  - 250- $\mu$ A / MIPS active
- High-performance analog ideal for precision measurement
  - 12-bit or 10-bit ADC — 200 ksps, temperature sensor,  $V_{Ref}$
  - 12-bit dual-DAC
  - Comparator-gated timers for measuring resistive elements
  - Supply voltage supervisor
- 16-bit RISC CPU enables new applications at a fraction of the code size.
  - Large register file eliminates working file bottleneck
  - Compact core design reduces power consumption and cost
  - Optimized for modern high-level programming
  - Only 27 core instructions and seven addressing modes
  - Extensive vectored-interrupt capability
- In-system programmable Flash permits flexible code changes, field upgrades and data logging

## 1.2 Flexible Clock System

The clock system is designed specifically for battery-powered applications. A low-frequency auxiliary clock (ACLK) is driven directly from a common 32-kHz watch crystal. The ACLK can be used for a background real-time clock self wake-up function. An integrated high-speed digitally controlled oscillator (DCO) can source the master clock (MCLK) used by the CPU and high-speed peripherals. By design, the DCO is active and stable in less than 6  $\mu$ s. MSP430-based solutions effectively use the high-performance 16-bit RISC CPU in very short bursts.

- Low-frequency auxiliary clock = Ultralow-power stand-by mode
- High-speed master clock = High performance signal processing

Figure 1–1. MSP430 Architecture



### 1.3 Embedded Emulation

Dedicated embedded emulation logic resides on the device itself and is accessed via JTAG using no additional system resources.

The benefits of embedded emulation include:

- Unobtrusive development and debug with full-speed execution, breakpoints, and single-steps in an application are supported.
- Development is in-system subject to the same characteristics as the final application.
- Mixed-signal integrity is preserved and not subject to cabling interference.

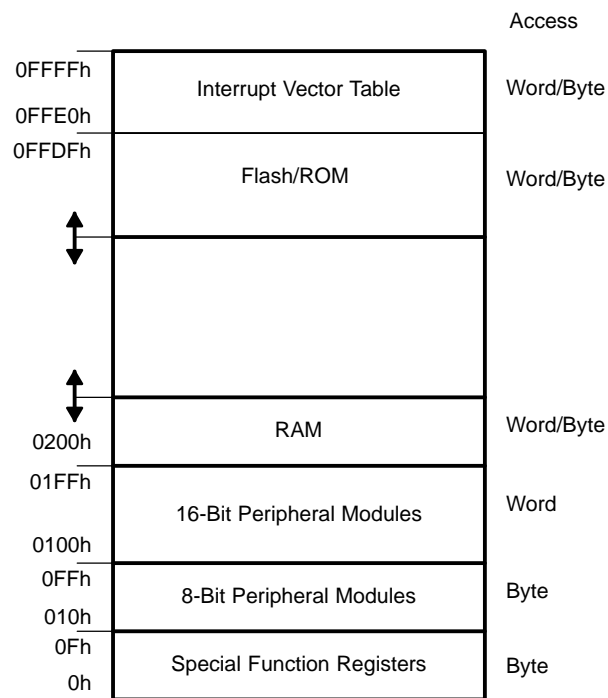


## 1.4 Address Space

The MSP430 von-Neumann architecture has one address space shared with special function registers (SFRs), peripherals, RAM, and Flash/ROM memory as shown in Figure 1–2. See the device-specific data sheets for specific memory maps. Code access are always performed on even addresses. Data can be accessed as bytes or words.

The addressable memory space is 64 KB with future expansion planned.

Figure 1–2. Memory Map



### 1.4.1 Flash/ROM

The start address of Flash/ROM depends on the amount of Flash/ROM present and varies by device. The end address for Flash/ROM is 0FFFFh. Flash can be used for both code and data. Word or byte tables can be stored and used in Flash/ROM without the need to copy the tables to RAM before using them.

The interrupt vector table is mapped into the upper 16 words of Flash/ROM address space, with the highest priority interrupt vector at the highest Flash/ROM word address (0FFFEh).

### 1.4.2 RAM

RAM starts at 0200h. The end address of RAM depends on the amount of RAM present and varies by device. RAM can be used for both code and data.

### 1.4.3 Peripheral Modules

Peripheral modules are mapped into the address space. The address space from 0100 to 01FFh is reserved for 16-bit peripheral modules. These modules should be accessed with word instructions. If byte instructions are used, only even addresses are permissible, and the high byte of the result is always 0.

The address space from 010h to 0FFh is reserved for 8-bit peripheral modules. These modules should be accessed with byte instructions. Read access of byte modules using word instructions results in unpredictable data in the high byte. If word data is written to a byte module only the low byte is written into the peripheral register, ignoring the high byte.

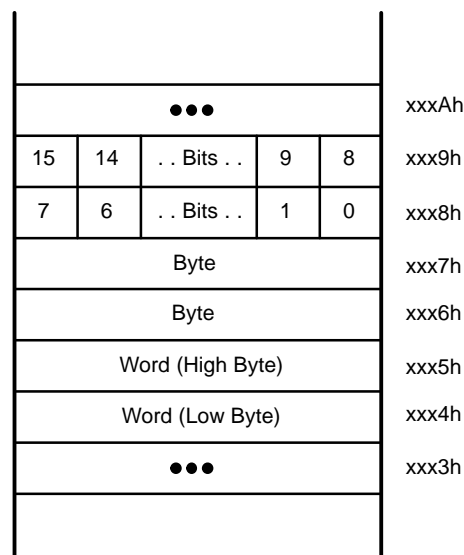
### 1.4.4 Special Function Registers (SFRs)

Some peripheral functions are configured in the SFRs. The SFRs are located in the lower 16 bytes of the address space, and are organized by byte. SFRs must be accessed using byte instructions only. See the device-specific data sheets for applicable SFR bits.

### 1.4.5 Memory Organization

Bytes are located at even or odd addresses. Words are only located at even addresses as shown in Figure 1–3. When using word instructions, only even addresses may be used. The low byte of a word is always an even address. The high byte is at the next odd address. For example, if a data word is located at address xxx4h, then the low byte of that data word is located at address xxx4h, and the high byte of that word is located at address xxx5h.

Figure 1–3. Bits, Bytes, and Words in a Byte-Organized Memory



# System Resets, Interrupts, and Operating Modes

---

---

---

---

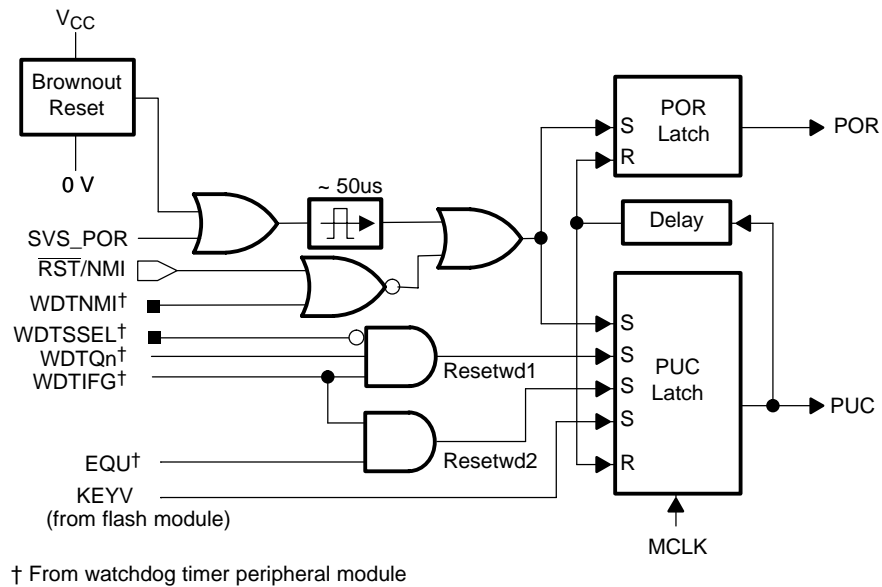
This chapter describes the MSP430x4xx system resets, interrupts, and operating modes.

<b>Topic</b>	<b>Page</b>
2.1 System Reset and Initialization .....	2-2
2.2 Interrupts .....	2-5
2.3 Operating Modes .....	2-13
2.4 Principles for Low-Power Applications .....	2-16
2.5 Connection of Unused Pins .....	2-16

## 2.1 System Reset and Initialization

The system reset circuitry shown in Figure 2–1 sources both a power-on reset (POR) and a power-up clear (PUC) signal. Different events trigger these reset signals and different initial conditions exist depending on which signal was generated.

Figure 2–1. Power-On Reset and Power-Up Clear Schematic



A POR is a device reset. A POR is only generated by the following three events:

- Powering up the device
- A low signal on the  $\overline{\text{RST/NMI}}$  pin when configured in the reset mode
- An SVS low condition when  $\text{PORON} = 1$ .

A PUC is always generated when a POR is generated, but a POR is not generated by a PUC. The following events trigger a PUC:

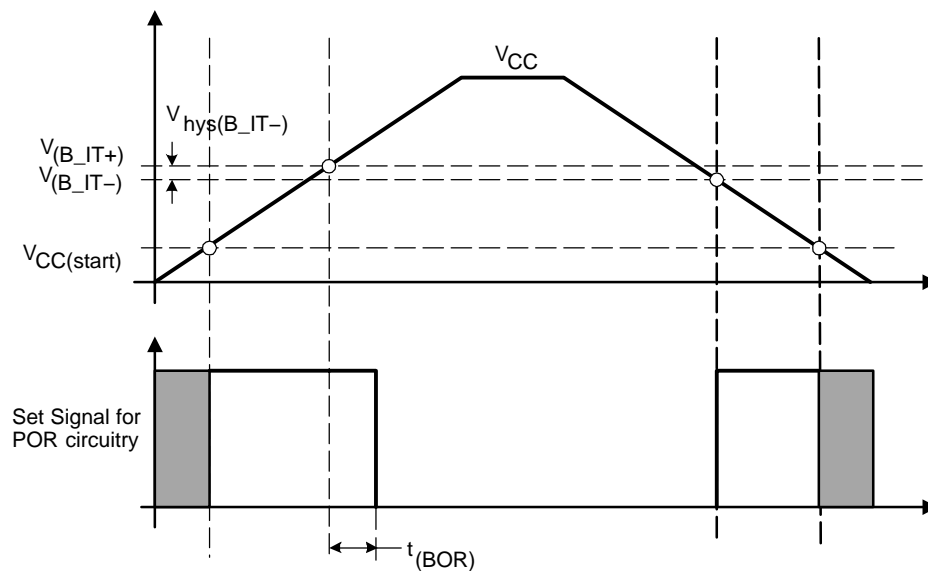
- A POR signal
- Watchdog timer expiration when in watchdog mode only
- Watchdog timer security key violation
- A Flash memory security key violation

### 2.1.1 Brownout Reset (BOR)

All MSP430x4xx devices have a brownout reset circuit. The brownout reset circuit detects low supply voltages such as when a supply voltage is applied to or removed from the  $V_{CC}$  terminal. The brownout reset circuit resets the device by triggering a POR signal when power is applied or removed. The operating levels are shown in Figure 2–2.

The POR signal becomes active when  $V_{CC}$  crosses the  $V_{CC(start)}$  level. It remains active until  $V_{CC}$  crosses the  $V_{(B\_IT+)}$  threshold and the delay  $t_{(BOR)}$  elapses. The delay  $t_{(BOR)}$  is adaptive being longer for a slow ramping  $V_{CC}$ . The hysteresis  $V_{hys(B\_IT-)}$  ensures that the supply voltage must drop below  $V_{(B\_IT-)}$  to generate another POR signal from the brownout reset circuitry.

Figure 2–2. Brownout Timing



As the  $V_{(B\_IT-)}$  level is significantly above the  $V_{(MIN)}$  level of the POR circuit, the BOR provides a reset for power failures where  $V_{CC}$  does not fall below  $V_{(MIN)}$ . See device-specific datasheet for parameters.

### 2.1.2 Device Initial Conditions After System Reset

After a POR, the initial MSP430 conditions are:

- The  $\overline{\text{RST}}$ /NMI pin is configured in the reset mode.
- I/O pins are switched to input mode as described in the *Digital I/O* chapter.
- Other peripheral modules and registers are initialized as described in their respective chapters in this manual.
- Status register (SR) is reset.
- The watchdog timer powers up active in watchdog mode.
- Program counter (PC) is loaded with address contained at reset vector location (0FFFEh). CPU execution begins at that address.

### Software Initialization

After a system reset, user software must initialize the MSP430 for the application requirements. The following must occur:

- Initialize the SP, typically to the top of RAM.
- Initialize the watchdog to the requirements of the application.
- Configure peripheral modules to the requirements of the application.

Additionally, the watchdog timer, oscillator fault, and flash memory flags can be evaluated to determine the source of the reset.

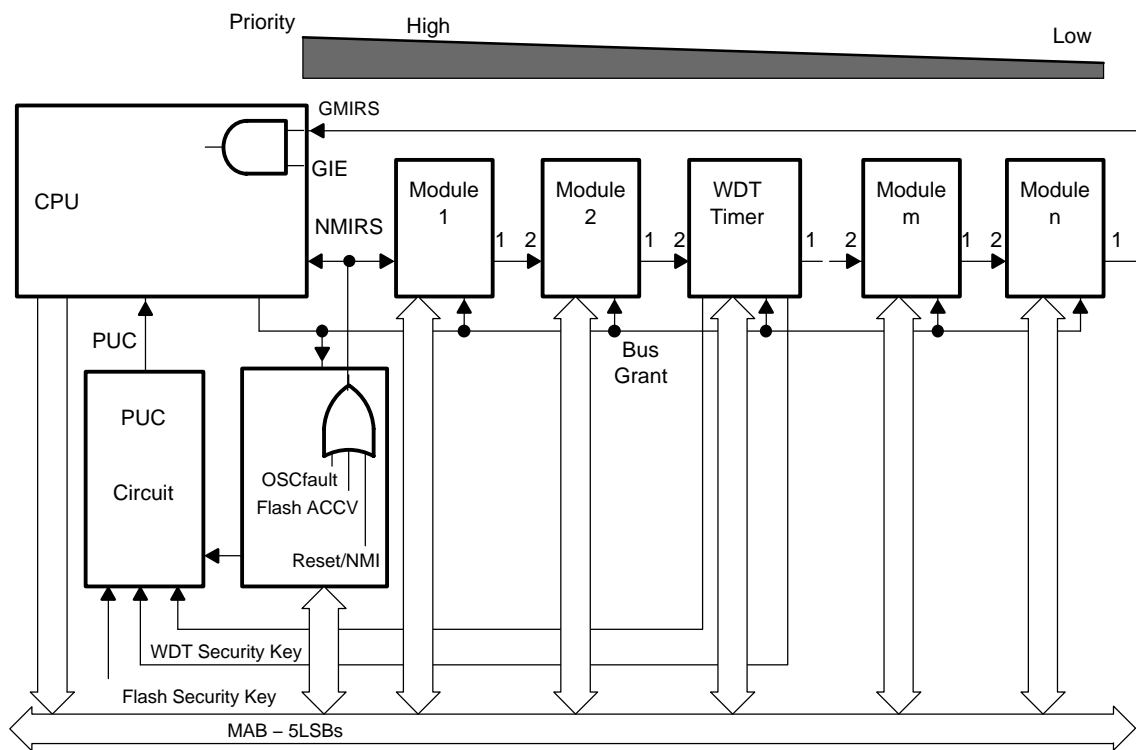
## 2.2 Interrupts

The interrupt priorities are fixed and defined by the arrangement of the modules in the connection chain as shown in Figure 2–3. The nearer a module is to the CPU/NMIRS, the higher the priority. Interrupt priorities determine what interrupt is taken when more than one interrupt is pending simultaneously.

There are three types of interrupts:

- System reset
- (Non)-maskable NMI
- Maskable

Figure 2–3. Interrupt Priority



## 2.2.1 (Non)-Maskable Interrupts (NMI)

(Non)-maskable NMI interrupts are not masked by the general interrupt enable bit (GIE), but are enabled by individual interrupt enable bits (ACCVIE, NMIIIE, OFIE). When a NMI interrupt is accepted, all NMI interrupt enable bits are automatically reset. Program execution begins at the address stored in the (non)-maskable interrupt vector, 0FFFCh. User software must set the required NMI interrupt enable bits for the interrupt to be re-enabled. The block diagram for NMI sources is shown in Figure 2–4.

A (non)-maskable NMI interrupt can be generated by three sources:

- An edge on the  $\overline{\text{RST}}/\text{NMI}$  pin when configured in NMI mode
- An oscillator fault occurs
- An access violation to the flash memory

### Reset/NMI Pin

At power-up, the  $\overline{\text{RST}}/\text{NMI}$  pin is configured in the reset mode. The function of the  $\overline{\text{RST}}/\text{NMI}$  pins is selected in the watchdog control register WDTCTL. If the  $\overline{\text{RST}}/\text{NMI}$  pin is set to the reset function, the CPU is held in the reset state as long as the  $\overline{\text{RST}}/\text{NMI}$  pin is held low. After the input changes to a high state, the CPU starts program execution at the word address stored in the reset vector, 0FFFEh.

If the  $\overline{\text{RST}}/\text{NMI}$  pin is configured by user software to the NMI function, a signal edge selected by the WDTNMIES bit generates an NMI interrupt if the NMIIIE bit is set. The  $\overline{\text{RST}}/\text{NMI}$  flag NMIIFG is also set.

---

**Note: Holding  $\overline{\text{RST}}/\text{NMI}$  Low**

When configured in the NMI mode, a signal generating an NMI event should not hold the  $\overline{\text{RST}}/\text{NMI}$  pin low. If a PUC occurs from a different source while the NMI signal is low, the device will be held in the reset state because a PUC changes the  $\overline{\text{RST}}/\text{NMI}$  pin to the reset function.

---

---

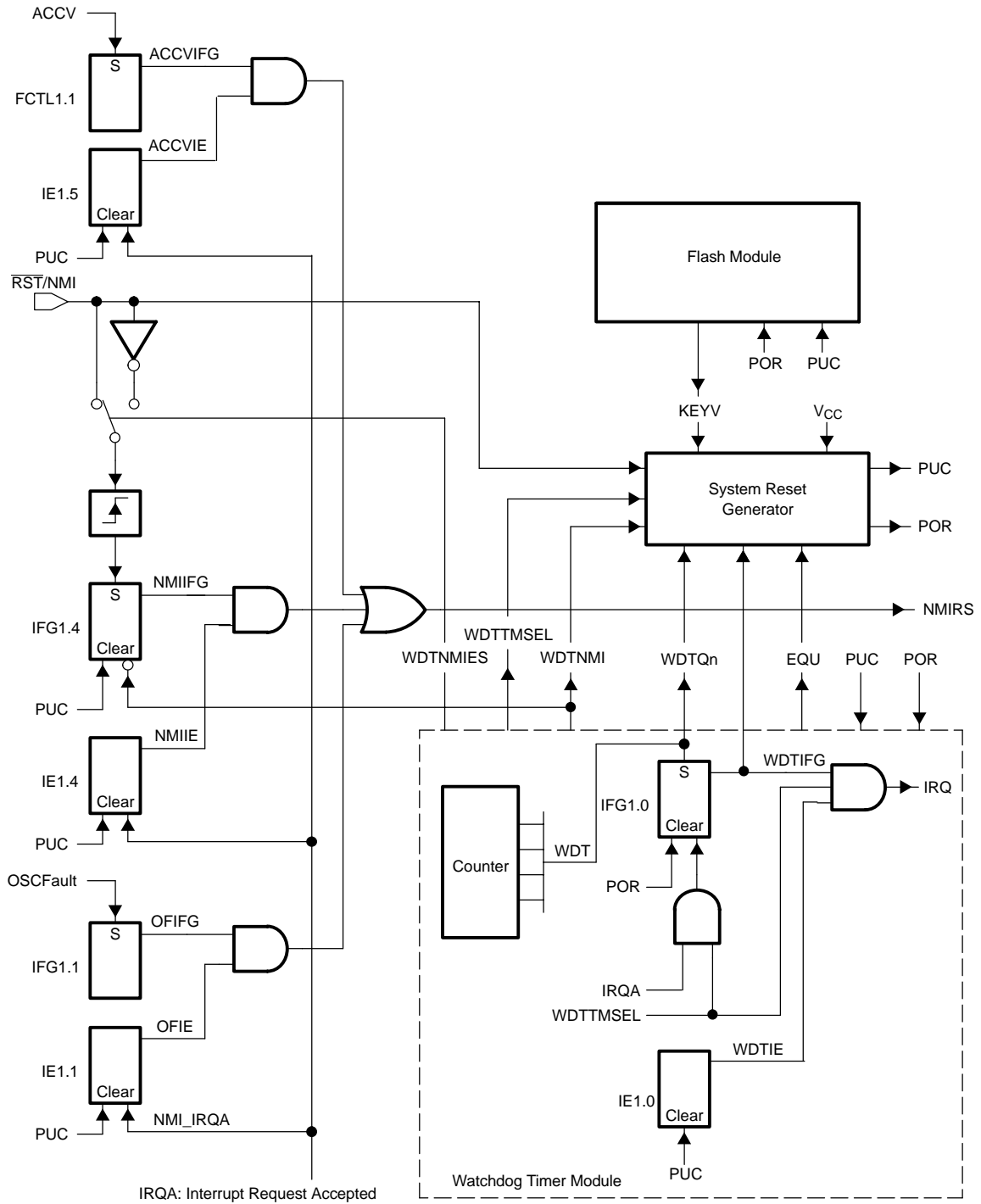
**Note: Modifying WDTNMIES**

When NMI mode is selected and the WDTNMIES bit is changed, an NMI can be generated, depending on the actual level at the  $\overline{\text{RST}}/\text{NMI}$  pin. When the NMI edge select bit is changed before selecting the NMI mode, no NMI is generated.

---



Figure 2–4. Block Diagram of (Non)-Maskable Interrupt Sources



## **Oscillator Fault**

The oscillator fault signal warns of a possible error condition with the crystal oscillator. The oscillator fault can be enabled to generate an NMI interrupt by setting the OFIE bit. The OFIFG flag can then be tested by NMI the interrupt service routine to determine if the NMI was caused by an oscillator fault.

A PUC signal can trigger an oscillator fault, because the PUC switches the LFXT1 to LF mode, therefore switching off the HF mode. The PUC signal also switches off the XT2 oscillator.

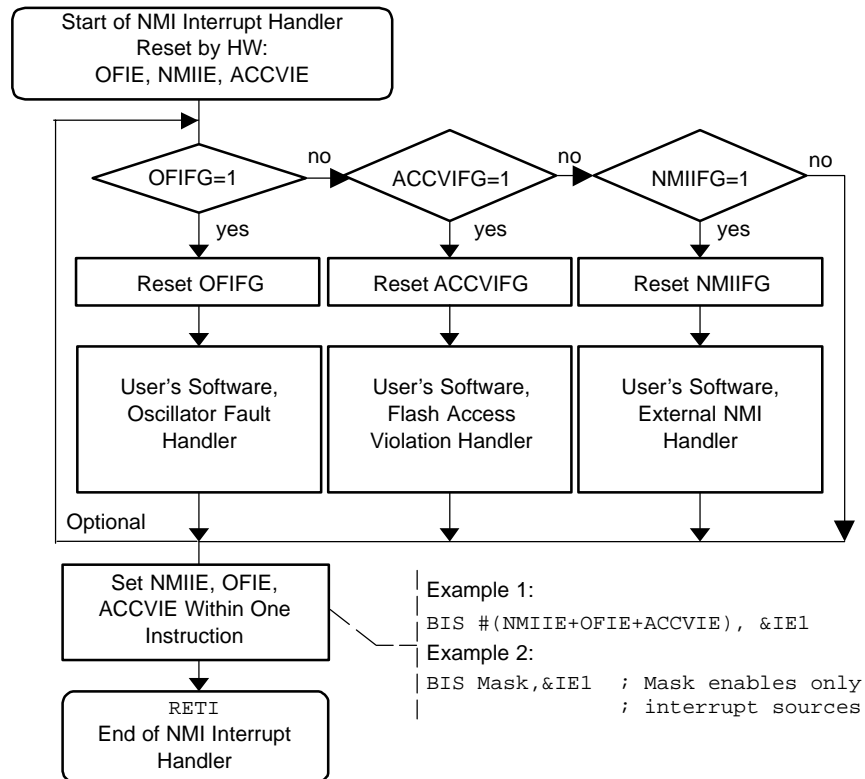
## **Flash Access Violation**

The flash ACCVIFG flag is set when a flash access violation occurs. The flash access violation can be enabled to generate an NMI interrupt by setting the ACCVIE bit. The ACCVIFG flag can then be tested by NMI the interrupt service routine to determine if the NMI was caused by a flash access violation.

### Example of an NMI Interrupt Handler

The NMI interrupt is a multiple-source interrupt. An NMI interrupt automatically resets the NMIIE, OFIE and ACCVIE interrupt-enable bits. The user NMI service routine resets the interrupt flags and re-enables the interrupt-enable bits according to the application needs as shown in Figure 2-5.

Figure 2-5. NMI Interrupt Handler



**Note: Enabling NMI Interrupts with ACCVIE, NMIIE, and OFIE**

Care should be taken when the ACCVIE, NMIIE, and OFIE enable bits are set inside of an NMI interrupt service routine. This re-enables the interrupt and can cause stack overflow if the interrupt flag has become set, due to nested interrupts. When set inside of an NMI service routine, they should be set by the last instruction of the routine before the `RETI` instruction.

### 2.2.2 Maskable Interrupts

Maskable interrupts are caused by peripherals with interrupt capability including the watchdog timer overflow in interval-timer mode. Each maskable interrupt source can be disabled individually by an interrupt enable bit, or all maskable interrupts can be disabled by the general interrupt enable (GIE) bit in the status register (SR).

Each individual peripheral interrupt is discussed in the associated peripheral module chapter in this manual.

### 2.2.3 Interrupt Processing

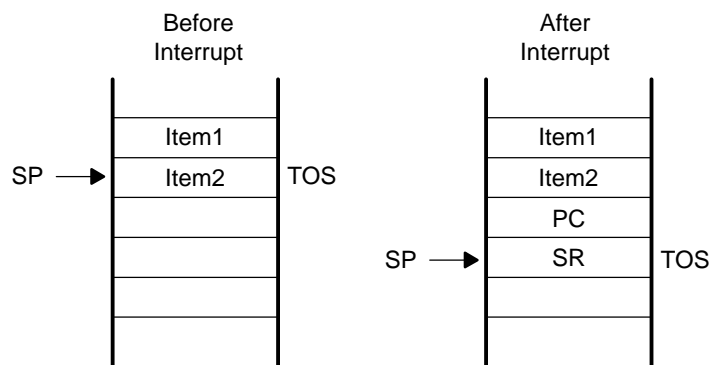
When an interrupt is requested from a peripheral and the peripheral interrupt enable bit and GIE bit are set, the interrupt service routine is requested. Only the individual enable bit must be set for (non)-maskable interrupts to be requested.

#### Interrupt Acceptance

The interrupt latency is 6 cycles, starting with the acceptance of an interrupt request, and lasting until the start of execution of the first instruction of the interrupt-service routine, as shown in Figure 2–6. The interrupt logic executes the following:

- 1) Any currently executing instruction is completed.
- 2) The PC, which points to the next instruction, is pushed onto the stack.
- 3) The SR is pushed onto the stack.
- 4) The interrupt with the highest priority is selected if multiple interrupts occurred during the last instruction and are pending for service.
- 5) The interrupt request flag resets automatically on single-source flags. Multiple source flags remain set for servicing by software.
- 6) The SR is cleared with the exception of SCG0, which is left unchanged. This terminates any low-power mode. Because the GIE bit is cleared, further interrupts are disabled.
- 7) The content of the interrupt vector is loaded into the PC: the program continues with the interrupt service routine at that address.

Figure 2–6. Interrupt Processing



## Return From Interrupt

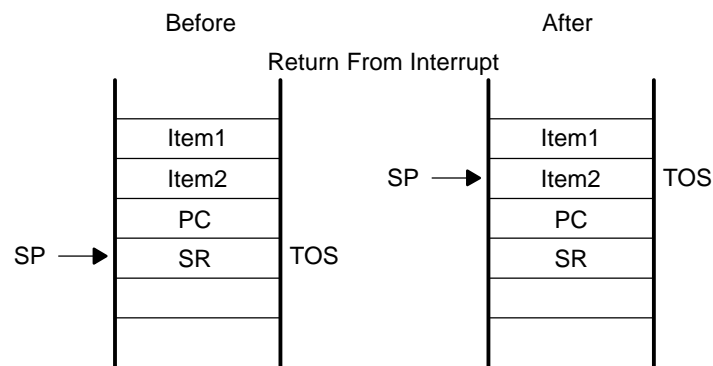
The interrupt handling routine terminates with the instruction:

`RETI` (return from an interrupt service routine)

The return from the interrupt takes 5 cycles to execute the following actions and is illustrated in Figure 2–7.

- 1) The SR with all previous settings pops from the stack. All previous settings of GIE, CPUOFF, etc. are now in effect, regardless of the settings used during the interrupt service routine.
- 2) The PC pops from the stack and begins execution at the point where it was interrupted.

Figure 2–7. Return From Interrupt



Interrupt nesting is enabled if the GIE bit is set inside an interrupt service routine. When interrupt nesting is enabled, any interrupt occurring during an interrupt service routine will interrupt the routine, regardless of the interrupt priorities.

## 2.2.4 Interrupt Vectors

The interrupt vectors and the power-up starting address are located in the address range 0FFFFh – 0FFE0h as described in Table 2–1. A vector is programmed by the user with the 16-bit address of the corresponding interrupt service routine. See the device-specific data sheet for the complete interrupt vector list.

Table 2–1. Interrupt Sources, Flags, and Vectors

INTERRUPT SOURCE	INTERRUPT FLAG	SYSTEM INTERRUPT	WORD ADDRESS	PRIORITY
Power-up, external reset, watchdog, flash password	WDTIFG KEYV	Reset	0FFFEh	15, highest
NMI, oscillator fault, flash memory access violation	NMIIFG OFIFG ACCVIFG	(non)-maskable (non)-maskable (non)-maskable	0FFFCh	14
device-specific			0FFFAh	13
device-specific			0FFF8h	12
device-specific			0FFF6h	11
Watchdog timer	WDTIFG	maskable	0FFF4h	10
device-specific			0FFF2h	9
device-specific			0FFF0h	8
device-specific			0FFEEh	7
device-specific			0FFECCh	6
device-specific			0FFEAh	5
device-specific			0FFE8h	4
device-specific			0FFE6h	3
device-specific			0FFE4h	2
device-specific			0FFE2h	1
device-specific			0FFE0h	0, lowest

## 2.2.5 Special Function Registers (SFRs)

Some module enable bits, interrupt enable bits, and interrupt flags are located in the SFRs. The SFRs are located in the lower address range and are implemented in byte format. SFRs must be accessed using byte instructions. See the device-specific datasheet for the SFR configuration.

## 2.3 Operating Modes

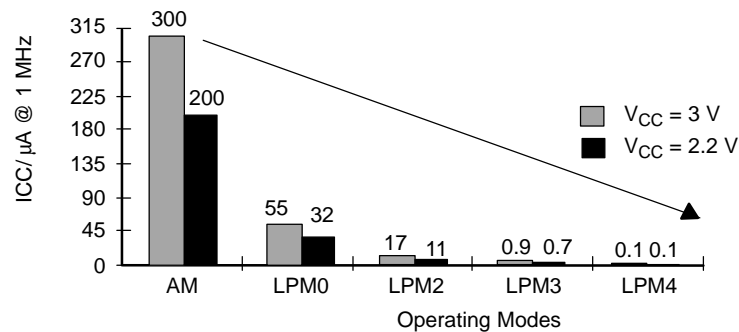
The MSP430 family is designed for ultralow-power applications and uses different operating modes shown in Figure 2–9.

The operating modes take into account three different needs:

- Ultralow-power
- Speed and data throughput
- Minimization of individual peripheral current consumption

The MSP430 typical current consumption is shown in Figure 2–8.

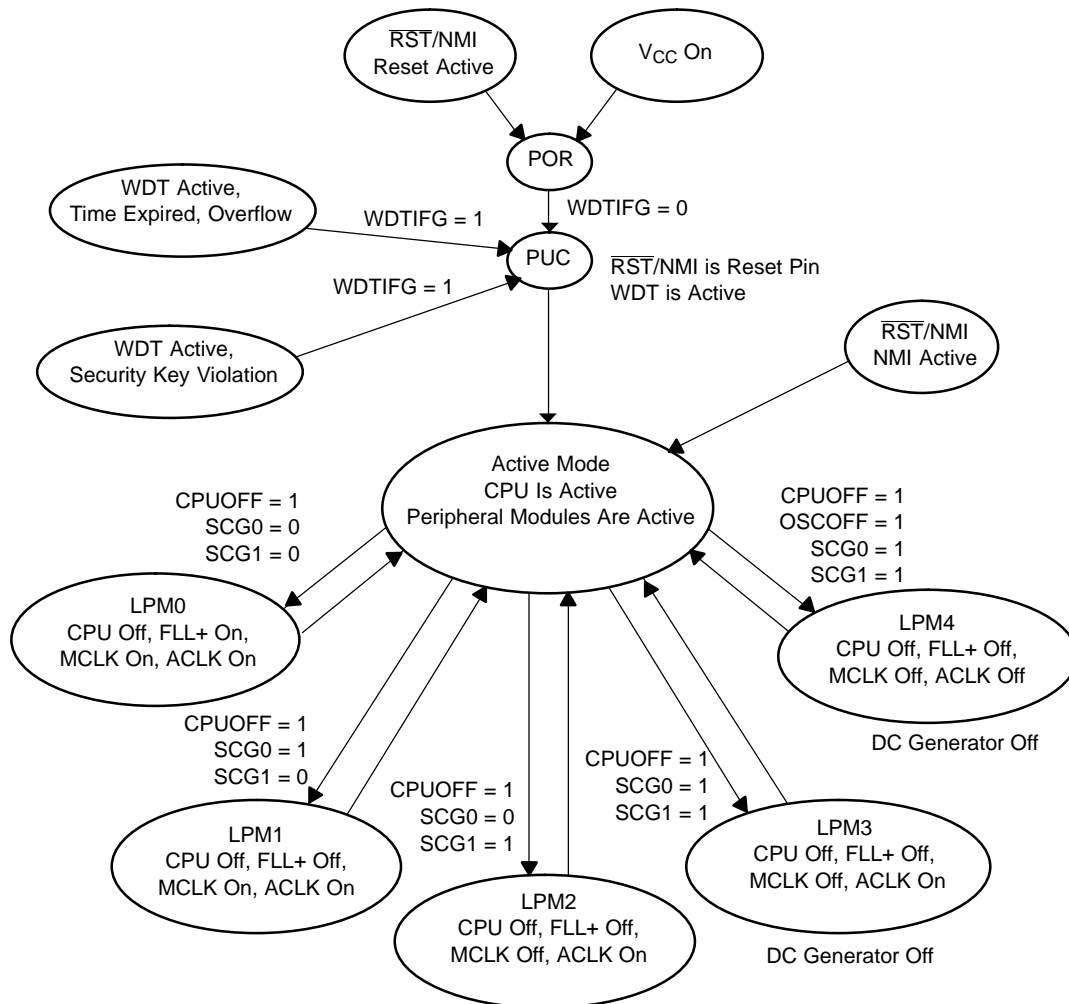
Figure 2–8. Typical Current Consumption of 41x Devices vs Operating Modes



The low-power modes 0–4 are configured with the CPUOFF, OSCOFF, SCG0, and SCG1 bits in the status register. The advantage of including the CPUOFF, OSCOFF, SCG0, and SCG1 mode-control bits in the status register is that the present operating mode is saved onto the stack during an interrupt service routine. Program flow returns to the previous operating mode if the saved SR value is not altered during the interrupt service routine. Program flow can be returned to a different operating mode by manipulating the saved SR value on the stack inside of the interrupt service routine. The mode-control bits and the stack can be accessed with any instruction.

When setting any of the mode-control bits, the selected operating mode takes effect immediately. Peripherals operating with any disabled clock are disabled until the clock becomes active. The peripherals may also be disabled with their individual control register settings. All I/O port pins and RAM/registers are unchanged. Wake up is possible through all enabled interrupts.

Figure 2–9. MSP430x4xx Operating Modes For Basic Clock System



SCG1	SCG0	OSCOFF	CPUOFF	Mode	CPU and Clocks Status
0	0	0	0	Active	CPU is active, all enabled clocks are active
0	0	0	1	LPM0	CPU, MCLK are disabled SMCLK, ACLK are active
0	1	0	1	LPM1	CPU, MCLK, DCO osc. are disabled DC generator is disabled if the DCO is not used for MCLK or SMCLK in active mode SMCLK, ACLK are active
1	0	0	1	LPM2	CPU, MCLK, SMCLK, DCO osc. are disabled DC generator remains enabled ACLK is active
1	1	0	1	LPM3	CPU, MCLK, SMCLK, DCO osc. are disabled DC generator disabled ACLK is active
1	1	1	1	LPM4	CPU and all clocks disabled



### 2.3.1 Entering and Exiting Low-Power Modes

An enabled interrupt event wakes the MSP430 from any of the low-power operating modes. The program flow is:

- Enter interrupt service routine:
  - The PC and SR are stored on the stack
  - The CPUOFF, SCG1, and OSCOFF bits are automatically reset
- Options for returning from the interrupt service routine:
  - The original SR is popped from the stack, restoring the previous operating mode.
  - The SR bits stored on the stack can be modified within the interrupt service routine returning to a different operating mode when the RETI instruction is executed.

```

; Enter LPM0 Example
  BIS    #GIE+CPUOFF,SR          ; Enter LPM0
; ...                            ; Program stops here
;
; Exit LPM0 Interrupt Service Routine
  BIC    #CPUOFF,0(SP)          ; Exit LPM0 on RETI
  RETI

; Enter LPM3 Example
  BIS    #GIE+CPUOFF+SCG1+SCG0,SR ; Enter LPM3
; ...                            ; Program stops here
;
; Exit LPM3 Interrupt Service Routine
  BIC    #CPUOFF+SCG1+SCG0,0(SP) ; Exit LPM3 on RETI
  RETI

```

### Extended Time in Low-Power Modes

The negative temperature coefficient of the DCO should be considered when the DCO is disabled for extended low-power mode periods. If the temperature changes significantly, the DCO frequency at wake-up may be significantly different from when the low-power mode was entered and may be out of the specified operating range. To avoid this, the DCO can be set to its lowest value before entering the low-power mode for extended periods of time where temperature can change.

```

; Enter LPM4 Example with lowest DCO Setting
  BIC.B  #FN_8+FN_4+FN_3+FN_2,&SCFI0 ; Lowest Range
  MOV.B  #010h,&SCFI1                ; Select Tap 2
  BIS    #GIE+CPUOFF+OSCOFF+SCG1+SCG0,SR ; Enter LPM4
; ...                            ; Program stops
;
; Interrupt Service Routine
  BIC    #CPUOFF+OSCOFF+SCG1+SCG0,0(SR); Exit LPM4 on RETI
  RETI

```

## 2.4 Principles for Low-Power Applications

Often, the most important factor for reducing power consumption is using the MSP430's clock system to maximize the time in LPM3. LPM3 power consumption is less than 2  $\mu$ A typical with both a real-time clock function and all interrupts active. A 32-kHz watch crystal is used for the ACLK and the CPU is clocked from the DCO (normally off) which has a 6- $\mu$ s wake-up.

- Use interrupts to wake the processor and control program flow.
- Peripherals should be switched on only when needed.
- Use low-power integrated peripheral modules in place of software driven functions. For example Timer\_A and Timer\_B can automatically generate PWM and capture external timing, with no CPU resources.
- Calculated branching and fast table look-ups should be used in place of flag polling and long software calculations.
- Avoid frequent subroutine and function calls due to overhead.
- For longer software routines, single-cycle CPU registers should be used.

## 2.5 Connection of Unused Pins

The correct termination of all unused pins is listed in Table 2–2.

Table 2–2. Connection of Unused Pins

Pin	Potential	Comment
<b>AV<sub>CC</sub></b>	DV <sub>CC</sub>	
<b>AV<sub>SS</sub></b>	DV <sub>SS</sub>	
<b>V<sub>REF+</sub></b>	Open	
<b>Ve<sub>REF+</sub></b>	DV <sub>SS</sub>	
<b>V<sub>REF-</sub>/Ve<sub>REF-</sub></b>	DV <sub>SS</sub>	
<b>XIN</b>	DV <sub>CC</sub>	
<b>XOUT</b>	Open	
<b>XT2IN</b>	DV <sub>SS</sub>	43x and 44x devices
<b>XT2OUT</b>	Open	43x and 44x devices
<b>Px.0 to Px.7</b>	Open	Switched to port function, output direction
<b>R<sub>ST</sub>/NMI</b>	DV <sub>CC</sub> or V <sub>CC</sub>	47 k $\Omega$ pullup with 10nF pull down
<b>R03</b>	DV <sub>SS</sub>	
<b>COM0</b>	Open	
<b>TDO</b>	Open	
<b>TDI</b>	Open	
<b>TMS</b>	Open	
<b>TCK</b>	Open	
<b>Sxx</b>	Open	

# RISC 16-Bit CPU

---

---

---

---

This chapter describes the MSP430 CPU, addressing modes, and instruction set.

<b>Topic</b>	<b>Page</b>
<b>3.1 CPU Introduction</b> .....	<b>3-2</b>
<b>3.2 CPU Registers</b> .....	<b>3-4</b>
<b>3.3 Addressing Modes</b> .....	<b>3-9</b>
<b>3.4 Instruction Set</b> .....	<b>3-17</b>

### 3.1 CPU Introduction

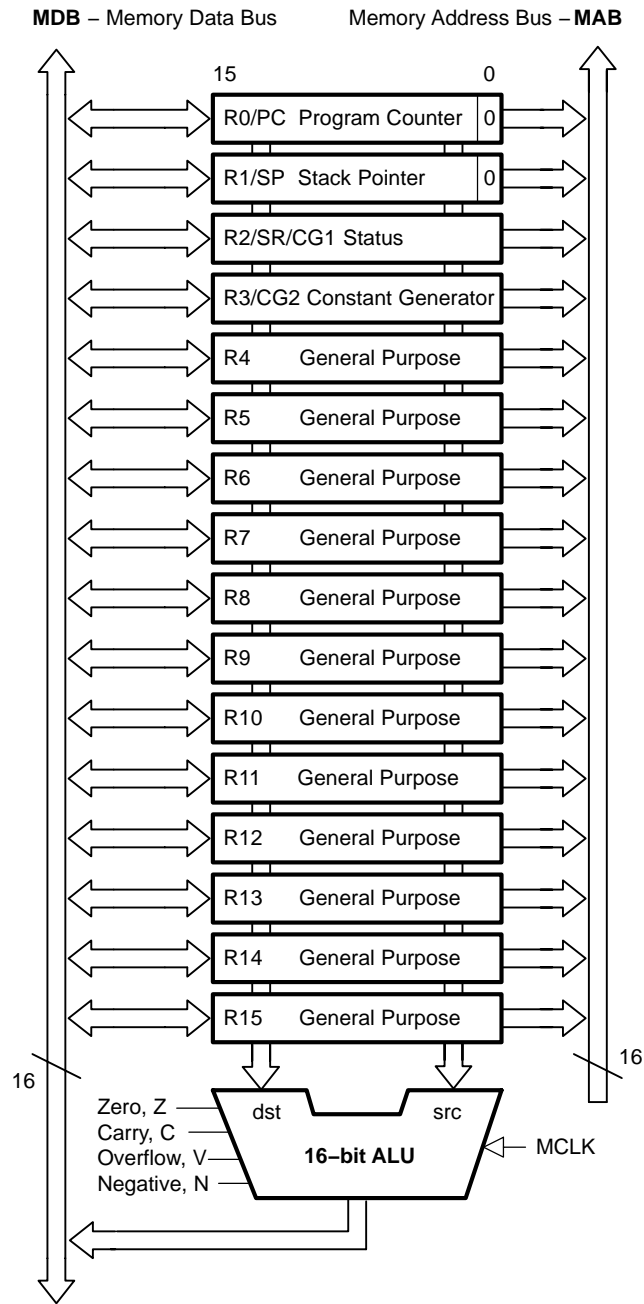
The CPU incorporates features specifically designed for modern programming techniques such as calculated branching, table processing and the use of high-level languages such as C. The CPU can address the complete address range without paging.

The CPU features include:

- RISC architecture with 27 instructions and 7 addressing modes.
- Orthogonal architecture with every instruction usable with every addressing mode.
- Full register access including program counter, status registers, and stack pointer.
- Single-cycle register operations.
- Large 16-bit register file reduces fetches to memory.
- 16-bit address bus allows direct access and branching throughout entire memory range.
- 16-bit data bus allows direct manipulation of word-wide arguments.
- Constant generator provides six most used immediate values and reduces code size.
- Direct memory-to-memory transfers without intermediate register holding.
- Word and byte addressing and instruction formats.

The block diagram of the CPU is shown in Figure 3–1.

Figure 3-1. CPU Block Diagram



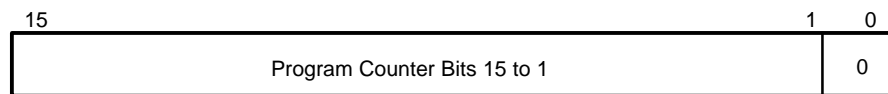
## 3.2 CPU Registers

The CPU incorporates sixteen 16-bit registers. R0, R1, R2 and R3 have dedicated functions. R4 to R15 are working registers for general use.

### 3.2.1 Program Counter (PC)

The 16-bit program counter (PC/R0) points to the next instruction to be executed. Each instruction uses an even number of bytes (two, four, or six), and the PC is incremented accordingly. Instruction accesses in the 64-KB address space are performed on word boundaries, and the PC is aligned to even addresses. Figure 3–2 shows the program counter.

Figure 3–2. Program Counter



The PC can be addressed with all instructions and addressing modes. A few examples:

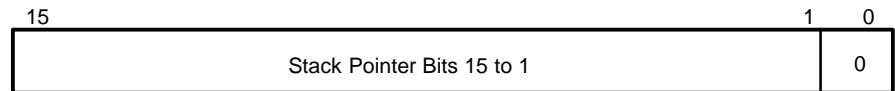
```
MOV    #LABEL,PC ; Branch to address LABEL
MOV    LABEL,PC  ; Branch to address contained in LABEL
MOV    @R14,PC   ; Branch indirect to address in R14
```

### 3.2.2 Stack Pointer (SP)

The stack pointer (SP/R1) is used by the CPU to store the return addresses of subroutine calls and interrupts. It uses a predecrement, postincrement scheme. In addition, the SP can be used by software with all instructions and addressing modes. Figure 3–3 shows the SP. The SP is initialized into RAM by the user, and is aligned to even addresses.

Figure 3–4 shows stack usage.

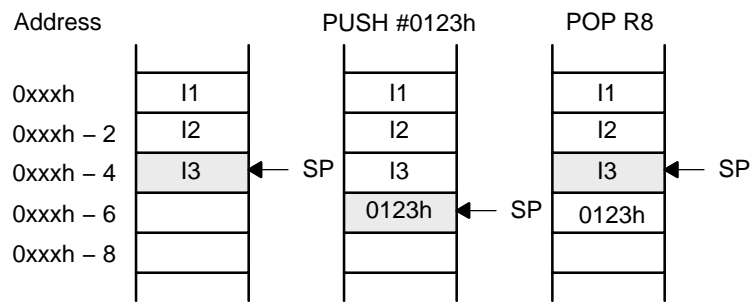
Figure 3–3. Stack Pointer



```

MOV    2(SP),R6 ; Item I2 -> R6
MOV    R7,0(SP) ; Overwrite TOS with R7
PUSH  #0123h   ; Put 0123h onto TOS
POP    R8      ; R8 = 0123h
    
```

Figure 3–4. Stack Usage



The special cases of using the SP as an argument to the PUSH and POP instructions are described and shown in Figure 3–5.

Figure 3–5. PUSH SP - POP SP Sequence



The stack pointer is changed after a PUSH SP instruction.

The stack pointer is not changed after a POP SP instruction. The POP SP instruction places SP1 into the stack pointer SP (SP2=SP1)

### 3.2.3 Status Register (SR)

The status register (SR/R2), used as a source or destination register, can be used in the register mode only addressed with word instructions. The remaining combinations of addressing modes are used to support the constant generator. Figure 3–6 shows the SR bits.

Figure 3–6. Status Register Bits

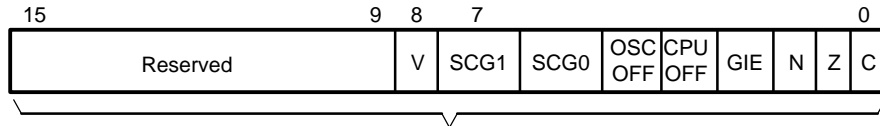


Table 3–1 describes the status register bits.

Table 3–1. Description of Status Register Bits

Bit	Description
V	<p>Overflow bit. This bit is set when the result of an arithmetic operation overflows the signed-variable range.</p> <p>ADD( .B ), ADDC( .B)                      Set when: Positive + Positive = Negative Negative + Negative = Positive, otherwise reset</p> <p>SUB( .B ), SUBC( .B ), CMP( .B)        Set when: Positive – Negative = Negative Negative – Positive = Positive, otherwise reset</p>
SCG1	System clock generator 1. This bit, when set, turns off the DCO dc generator, if DCOCLK is not used for MCLK or SMCLK.
SCG0	System clock generator 0. This bit, when set, turns off the FLL+ loop control
OSCOFF	Oscillator Off. This bit, when set, turns off the LFXT1 crystal oscillator, when LFXT1CLK is not use for MCLK or SMCLK
CPUOFF	CPU off. This bit, when set, turns off the CPU.
GIE	General interrupt enable. This bit, when set, enables maskable interrupts. When reset, all maskable interrupts are disabled.
N	<p>Negative bit. This bit is set when the result of a byte or word operation is negative and cleared when the result is not negative.</p> <p>Word operation:                              N is set to the value of bit 15 of the result</p> <p>Byte operation:                                N is set to the value of bit 7 of the result</p>
Z	Zero bit. This bit is set when the result of a byte or word operation is 0 and cleared when the result is not 0.
C	Carry bit. This bit is set when the result of a byte or word operation produced a carry and cleared when no carry occurred.



### 3.2.4 Constant Generator Registers CG1 and CG2

Six commonly-used constants are generated with the constant generator registers R2 and R3, without requiring an additional 16-bit word of program code. The constants are selected with the source-register addressing modes (As), as described in Table 3–2.

Table 3–2. Values of Constant Generators CG1, CG2

Register	As	Constant	Remarks
R2	00	-----	Register mode
R2	01	(0)	Absolute address mode
R2	10	00004h	+4, bit processing
R2	11	00008h	+8, bit processing
R3	00	00000h	0, word processing
R3	01	00001h	+1
R3	10	00002h	+2, bit processing
R3	11	0FFFFh	-1, word processing

The constant generator advantages are:

- No special instructions required
- No additional code word for the six constants
- No code memory access required to retrieve the constant

The assembler uses the constant generator automatically if one of the six constants is used as an immediate source operand. Registers R2 and R3, used in the constant mode, cannot be addressed explicitly; they act as source-only registers.

#### Constant Generator – Expanded Instruction Set

The RISC instruction set of the MSP430 has only 27 instructions. However, the constant generator allows the MSP430 assembler to support 24 additional, emulated instructions. For example, the single-operand instruction:

```
CLR          dst
```

is emulated by the double-operand instruction with the same length:

```
MOV          R3, dst
```

where the #0 is replaced by the assembler, and R3 is used with As=00.

```
INC          dst
```

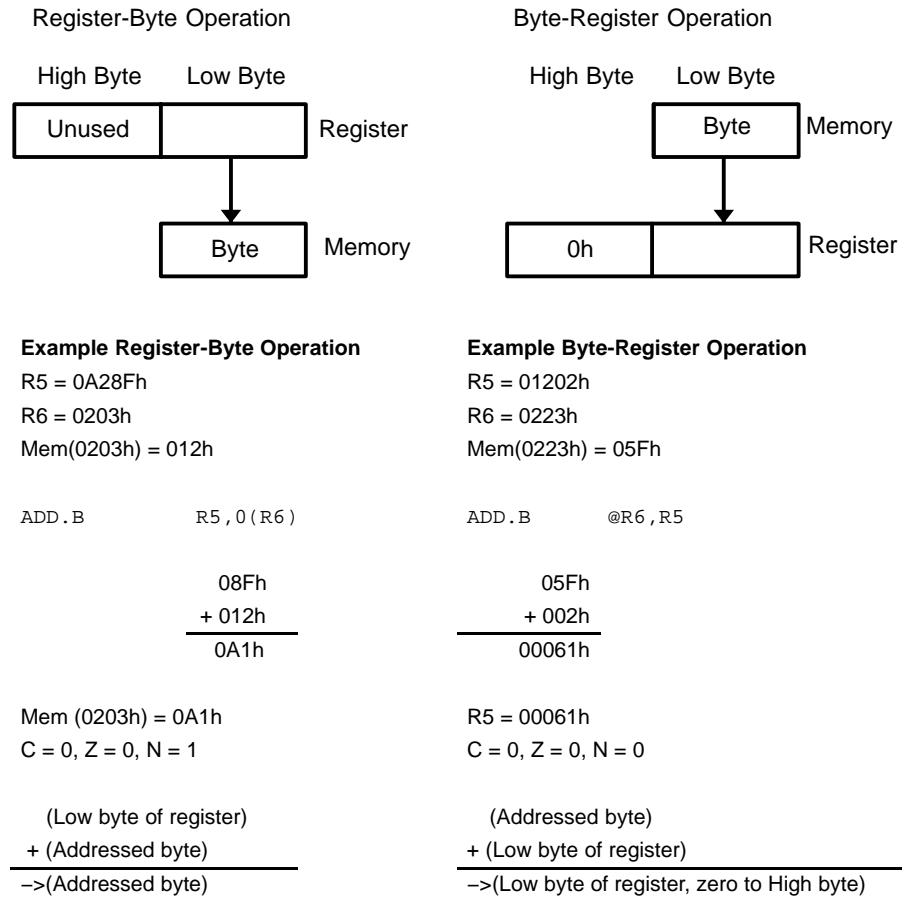
is replaced by:

```
ADD          0(R3), dst
```

### 3.2.5 General-Purpose Registers R4 - R15

The twelve registers, R4–R15, are general-purpose registers. All of these registers can be used as data registers, address pointers, or index values and can be accessed with byte or word instructions as shown in Figure 3–7.

Figure 3–7. Register-Byte/Byte-Register Operations



### 3.3 Addressing Modes

Seven addressing modes for the source operand and four addressing modes for the destination operand can address the complete address space with no exceptions. The bit numbers in Table 3–3 describe the contents of the As (source) and Ad (destination) mode bits.

Table 3–3. Source/Destination Operand Addressing Modes

As/Ad	Addressing Mode	Syntax	Description
00/0	Register mode	Rn	Register contents are operand
01/1	Indexed mode	X(Rn)	(Rn + X) points to the operand. X is stored in the next word.
01/1	Symbolic mode	ADDR	(PC + X) points to the operand. X is stored in the next word. Indexed mode X(PC) is used.
01/1	Absolute mode	&ADDR	The word following the instruction contains the absolute address. X is stored in the next word. Indexed mode X(SR) is used.
10/–	Indirect register mode	@Rn	Rn is used as a pointer to the operand.
11/–	Indirect autoincrement	@Rn+	Rn is used as a pointer to the operand. Rn is incremented afterwards by 1 for .B instructions and by 2 for .W instructions.
11/–	Immediate mode	#N	The word following the instruction contains the immediate constant N. Indirect autoincrement mode @PC+ is used.

The seven addressing modes are explained in detail in the following sections. Most of the examples show the same addressing mode for the source and destination, but any valid combination of source and destination addressing modes is possible in an instruction.

**Note: Use of Labels *EDE*, *TONI*, *TOM*, and *LEO***

Throughout MSP430 documentation *EDE*, *TONI*, *TOM*, and *LEO* are used as generic labels. They are only labels. They have no special meaning.

### 3.3.1 Register Mode

The register mode is described in Table 3–4.

*Table 3–4. Register Mode Description*

Assembler Code	Content of ROM
MOV R10,R11	MOV R10,R11

Length: One or two words

Operation: Move the content of R10 to R11. R10 is not affected.

Comment: Valid for source and destination

Example: MOV R10,R11

	Before:		After:
R10	<input type="text" value="0A023h"/>	R10	<input type="text" value="0A023h"/>
R11	<input type="text" value="0FA15h"/>	R11	<input type="text" value="0A023h"/>
PC	<input type="text" value="PC&lt;sub&gt;old&lt;/sub&gt;"/>	PC	<input type="text" value="PC&lt;sub&gt;old&lt;/sub&gt; + 2"/>

#### **Note: Data in Registers**

The data in the register can be accessed using word or byte instructions. If byte instructions are used, the high byte is always 0 in the result. The status bits are handled according to the result of the byte instruction.

### 3.3.2 Indexed Mode

The indexed mode is described in Table 3–5.

Table 3–5. Indexed Mode Description

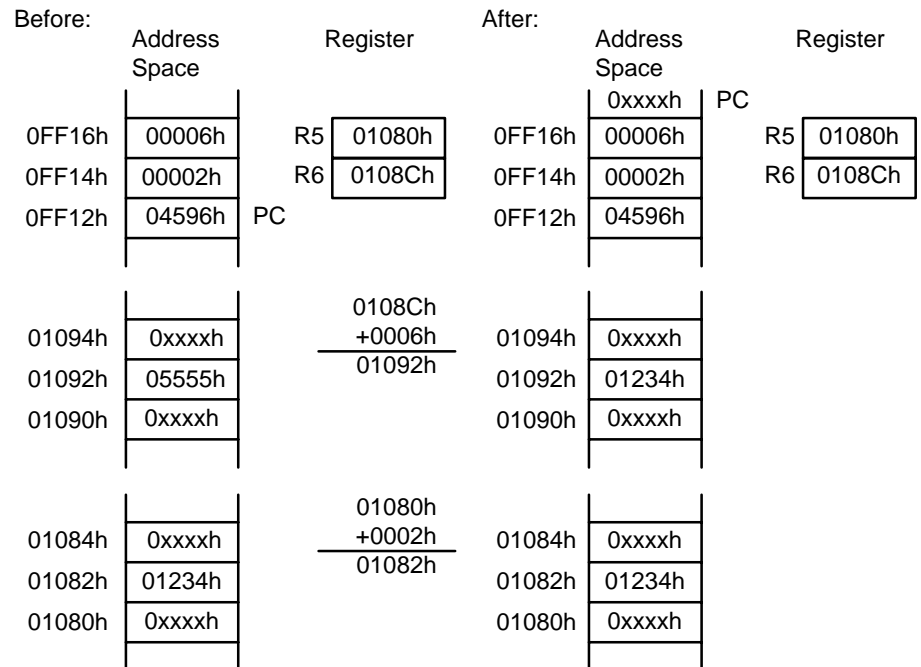
Assembler Code	Content of ROM
MOV 2(R5),6(R6)	MOV X(R5),Y(R6)
	X = 2
	Y = 6

Length: Two or three words

Operation: Move the contents of the source address (contents of R5 + 2) to the destination address (contents of R6 + 6). The source and destination registers (R5 and R6) are not affected. In indexed mode, the program counter is incremented automatically so that program execution continues with the next instruction.

Comment: Valid for source and destination

Example: MOV 2(R5),6(R6);



### 3.3.3 Symbolic Mode

The symbolic mode is described in Table 3–6.

Table 3–6. Symbolic Mode Description

Assembler Code	Content of ROM
MOV EDE,TONI	MOV X(PC),Y(PC) X = EDE - PC Y = TONI - PC

Length: Two or three words

Operation: Move the contents of the source address EDE (contents of PC + X) to the destination address TONI (contents of PC + Y). The words after the instruction contain the differences between the PC and the source or destination addresses. The assembler computes and inserts offsets X and Y automatically. With symbolic mode, the program counter (PC) is incremented automatically so that program execution continues with the next instruction.

Comment: Valid for source and destination

Example: MOV EDE,TONI ;Source address EDE = 0F016h  
;Dest. address TONI=01114h

Before:	Address Space	Register	After:	Address Space	Register
				0xxxxh	PC
0FF16h	011FEh		0FF16h	011FEh	
0FF14h	0F102h		0FF14h	0F102h	
0FF12h	04090h	PC	0FF12h	04090h	
0F018h	0xxxxh	0FF14h +0F102h ----- 0F016h	0F018h	0xxxxh	
0F016h	0A123h		0F016h	0A123h	
0F014h	0xxxxh		0F014h	0xxxxh	
01116h	0xxxxh	0FF16h +011FEh ----- 01114h	01116h	0xxxxh	
01114h	05555h		01114h	0A123h	
01112h	0xxxxh		01112h	0xxxxh	

### 3.3.4 Absolute Mode

The absolute mode is described in Table 3–7.

Table 3–7. Absolute Mode Description

Assembler Code	Content of ROM
MOV &EDE,&TONI	MOV X(0),Y(0) X = EDE Y = TONI

Length: Two or three words

Operation: Move the contents of the source address EDE to the destination address TONI. The words after the instruction contain the absolute address of the source and destination addresses. With absolute mode, the PC is incremented automatically so that program execution continues with the next instruction.

Comment: Valid for source and destination

Example: MOV &EDE,&TONI ;Source address EDE=0F016h,  
;dest. address TONI=01114h

Before:	Address Space	Register	After:	Address Space	Register
				0xxxxh	PC
0FF16h	01114h		0FF16h	01114h	
0FF14h	0F016h		0FF14h	0F016h	
0FF12h	04292h	PC	0FF12h	04292h	
0F018h	0xxxxh		0F018h	0xxxxh	
0F016h	0A123h		0F016h	0A123h	
0F014h	0xxxxh		0F014h	0xxxxh	
01116h	0xxxxh		01116h	0xxxxh	
01114h	01234h		01114h	0A123h	
01112h	0xxxxh		01112h	0xxxxh	

This address mode is mainly for hardware peripheral modules that are located at an absolute, fixed address. These are addressed with absolute mode to ensure software transportability (for example, position-independent code).

### 3.3.5 Indirect Register Mode

The indirect register mode is described in Table 3–8.

Table 3–8. Indirect Mode Description

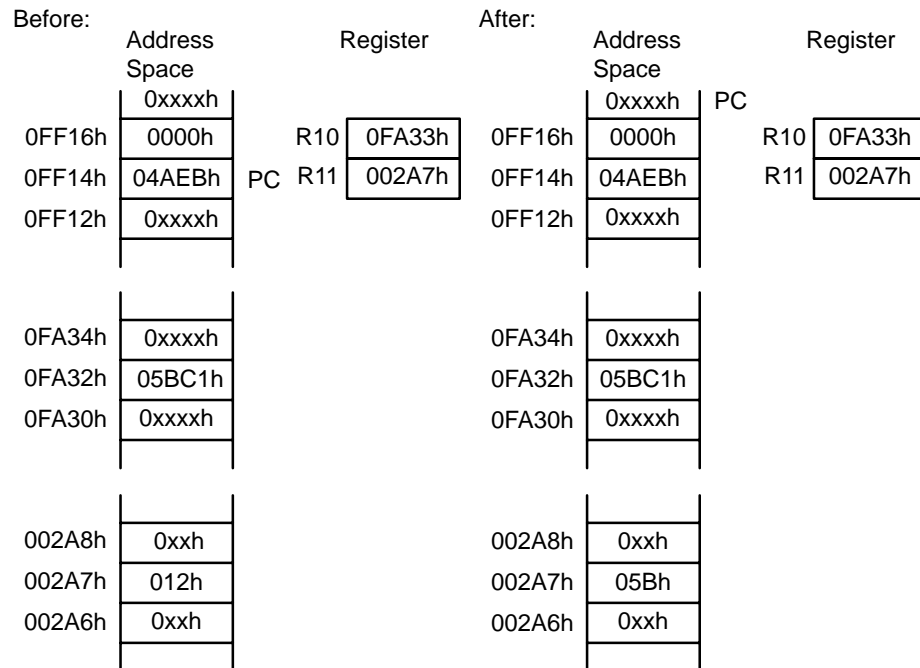
Assembler Code	Content of ROM
MOV @R10,0(R11)	MOV @R10,0(R11)

Length: One or two words

Operation: Move the contents of the source address (contents of R10) to the destination address (contents of R11). The registers are not modified.

Comment: Valid only for source operand. The substitute for destination operand is 0(Rd).

Example: MOV.B @R10,0(R11)





### 3.3.6 Indirect Autoincrement Mode

The indirect autoincrement mode is described in Table 3–9.

Table 3–9. Indirect Autoincrement Mode Description

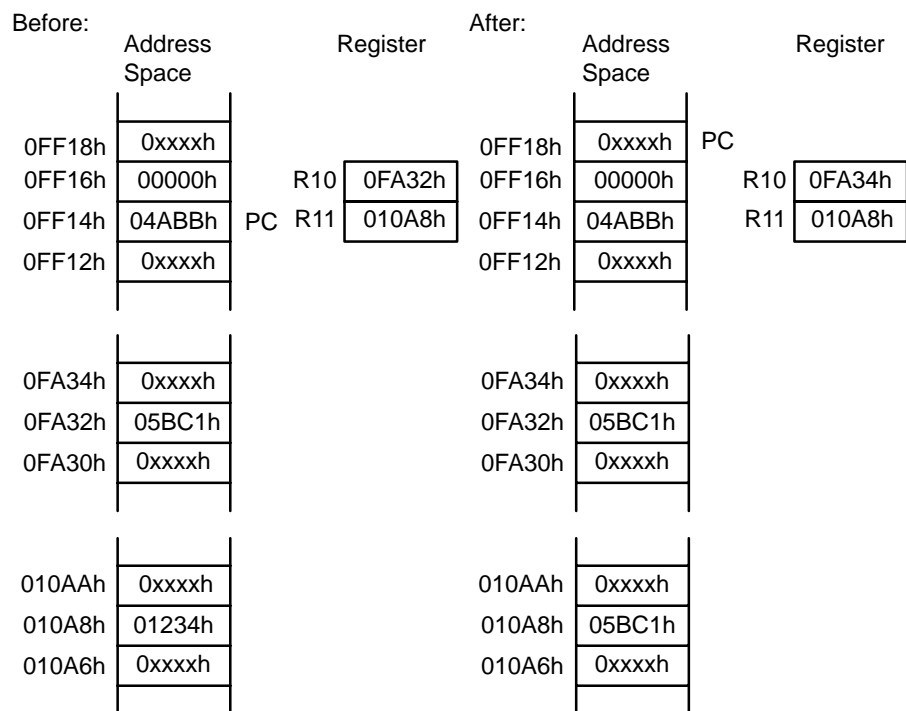
Assembler Code	Content of ROM
MOV @R10+,0(R11)	MOV @R10+,0(R11)

Length: One or two words

Operation: Move the contents of the source address (contents of R10) to the destination address (contents of R11). Register R10 is incremented by 1 for a byte operation, or 2 for a word operation after the fetch; it points to the next address without any overhead. This is useful for table processing.

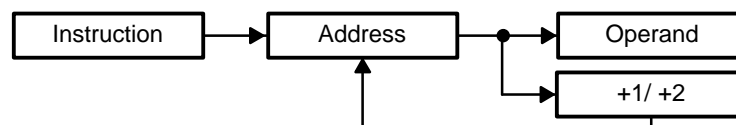
Comment: Valid only for source operand. The substitute for destination operand is 0(Rd) plus second instruction INCD Rd.

Example: MOV @R10+,0(R11)



The autoincrementing of the register contents occurs after the operand is fetched. This is shown in Figure 3–8.

Figure 3–8. Operand Fetch Operation



### 3.3.7 Immediate Mode

The immediate mode is described in Table 3–10.

Table 3–10. Immediate Mode Description

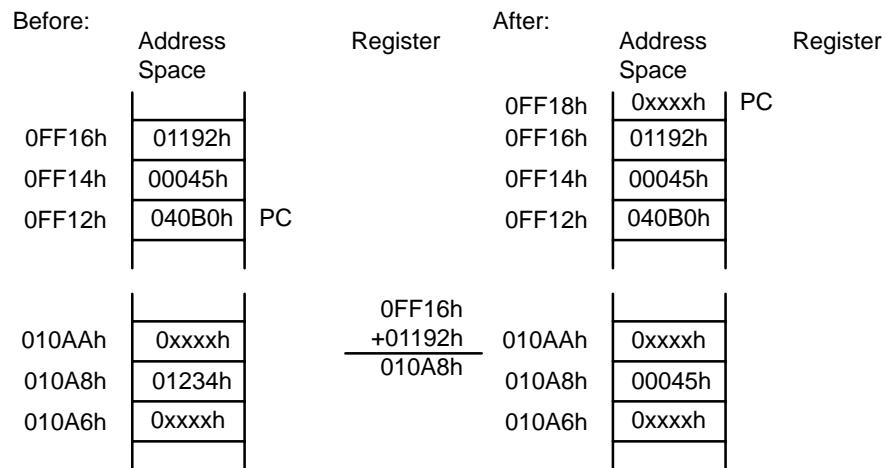
Assembler Code	Content of ROM
MOV #45h, TONI	MOV @PC+, X(PC) 45 X = TONI - PC

Length: Two or three words  
It is one word less if a constant of CG1 or CG2 can be used.

Operation: Move the immediate constant 45h, which is contained in the word following the instruction, to destination address TONI. When fetching the source, the program counter points to the word following the instruction and moves the contents to the destination.

Comment: Valid only for a source operand.

Example: MOV #45h, TONI



### 3.4 Instruction Set

The complete MSP430 instruction set consists of 27 core instructions and 24 emulated instructions. The core instructions are instructions that have unique op-codes decoded by the CPU. The emulated instructions are instructions that make code easier to write and read, but do not have op-codes themselves, instead they are replaced automatically by the assembler with an equivalent core instruction. There is no code or performance penalty for using emulated instruction.

There are three core-instruction formats:

- Dual-operand
- Single-operand
- Jump

All single-operand and dual-operand instructions can be byte or word instructions by using *.B* or *.W* extensions. Byte instructions are used to access byte data or byte peripherals. Word instructions are used to access word data or word peripherals. If no extension is used, the instruction is a word instruction.

The source and destination of an instruction are defined by the following fields:

src	The source operand defined by As and S-reg
dst	The destination operand defined by Ad and D-reg
As	The addressing bits responsible for the addressing mode used for the source (src)
S-reg	The working register used for the source (src)
Ad	The addressing bits responsible for the addressing mode used for the destination (dst)
D-reg	The working register used for the destination (dst)
B/W	Byte or word operation: 0: word operation 1: byte operation

---

**Note: Destination Address**

Destination addresses are valid anywhere in the memory map. However, when using an instruction that modifies the contents of the destination, the user must ensure the destination address is writable. For example, a masked-ROM location would be a valid destination address, but the contents are not modifiable, so the results of the instruction would be lost.

---

### 3.4.1 Double-Operand (Format I) Instructions

Figure 3–9 illustrates the double-operand instruction format.

Figure 3–9. Double Operand Instruction Format

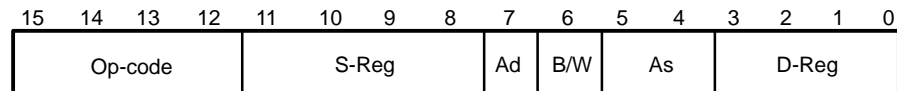


Table 3–11 lists and describes the double operand instructions.

Table 3–11. Double Operand Instructions

Mnemonic	S-Reg, D-Reg	Operation	Status Bits			
			V	N	Z	C
MOV( .B)	src, dst	src → dst	–	–	–	–
ADD( .B)	src, dst	src + dst → dst	*	*	*	*
ADDC( .B)	src, dst	src + dst + C → dst	*	*	*	*
SUB( .B)	src, dst	dst + .not.src + 1 → dst	*	*	*	*
SUBC( .B)	src, dst	dst + .not.src + C → dst	*	*	*	*
CMP( .B)	src, dst	dst – src	*	*	*	*
DADD( .B)	src, dst	src + dst + C → dst (decimally)	*	*	*	*
BIT( .B)	src, dst	src .and. dst	0	*	*	*
BIC( .B)	src, dst	.not.src .and. dst → dst	–	–	–	–
BIS( .B)	src, dst	src .or. dst → dst	–	–	–	–
XOR( .B)	src, dst	src .xor. dst → dst	*	*	*	*
AND( .B)	src, dst	src .and. dst → dst	0	*	*	*

- \* The status bit is affected
- The status bit is not affected
- 0 The status bit is cleared
- 1 The status bit is set

**Note: Instructions CMP and SUB**

The instructions CMP and SUB are identical except for the storage of the result. The same is true for the BIT and AND instructions.

### 3.4.2 Single-Operand (Format II) Instructions

Figure 3–10 illustrates the single-operand instruction format.

Figure 3–10. Single Operand Instruction Format

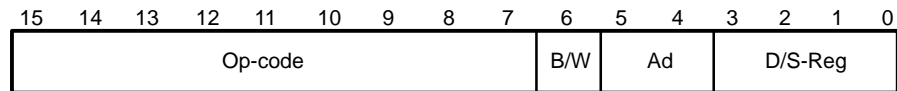


Table 3–12 lists and describes the single operand instructions.

Table 3–12. Single Operand Instructions

Mnemonic	S-Reg, D-Reg	Operation	Status Bits			
			V	N	Z	C
RRC (.B)	dst	C → MSB →.....LSB → C	*	*	*	*
RRA (.B)	dst	MSB → MSB →....LSB → C	0	*	*	*
PUSH (.B)	src	SP – 2 → SP, src → @SP	–	–	–	–
SWPB	dst	Swap bytes	–	–	–	–
CALL	dst	SP – 2 → SP, PC+2 → @SP	–	–	–	–
		dst → PC				
RETI		TOS → SR, SP + 2 → SP	*	*	*	*
		TOS → PC, SP + 2 → SP				
SXT	dst	Bit 7 → Bit 8.....Bit 15	0	*	*	*

- \* The status bit is affected
- The status bit is not affected
- 0 The status bit is cleared
- 1 The status bit is set

All addressing modes are possible for the `CALL` instruction. If the symbolic mode (`ADDRESS`), the immediate mode (`#N`), the absolute mode (`&EDE`) or the indexed mode `x(RN)` is used, the word that follows contains the address information.

### 3.4.3 Jumps

Figure 3–11 shows the conditional-jump instruction format.

Figure 3–11. Jump Instruction Format

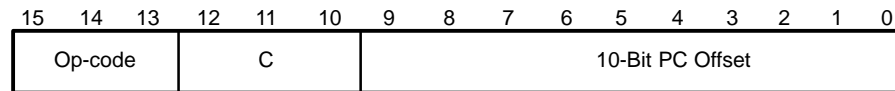


Table 3–13 lists and describes the jump instructions.

Table 3–13. Jump Instructions

Mnemonic	S-Reg, D-Reg	Operation
JEQ/JZ	Label	Jump to label if zero bit is set
JNE/JNZ	Label	Jump to label if zero bit is reset
JC	Label	Jump to label if carry bit is set
JNC	Label	Jump to label if carry bit is reset
JN	Label	Jump to label if negative bit is set
JGE	Label	Jump to label if (N .XOR. V) = 0
JL	Label	Jump to label if (N .XOR. V) = 1
JMP	Label	Jump to label unconditionally

Conditional jumps support program branching relative to the PC and do not affect the status bits. The possible jump range is from –511 to +512 words relative to the PC value at the jump instruction. The 10-bit program-counter offset is treated as a signed 10-bit value that is doubled and added to the program counter:

$$PC_{\text{new}} = PC_{\text{old}} + 2 + PC_{\text{offset}} \times 2$$

<b>ADC[.W]</b>	Add carry to destination
<b>ADC.B</b>	Add carry to destination
<b>Syntax</b>	ADC     dst    or   ADC.W   dst ADC.B   dst
<b>Operation</b>	dst + C → dst
<b>Emulation</b>	ADDC    #0,dst ADDC.B  #0,dst
<b>Description</b>	The carry bit (C) is added to the destination operand. The previous contents of the destination are lost.
<b>Status Bits</b>	N: Set if result is negative, reset if positive Z: Set if result is zero, reset otherwise C: Set if dst was incremented from 0FFFFh to 0000, reset otherwise Set if dst was incremented from 0FFh to 00, reset otherwise V: Set if an arithmetic overflow occurs, otherwise reset
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	The 16-bit counter pointed to by R13 is added to a 32-bit counter pointed to by R12. ADD        @R13,0(R12)     ; Add LSDs ADC        2(R12)           ; Add carry to MSD
<b>Example</b>	The 8-bit counter pointed to by R13 is added to a 16-bit counter pointed to by R12. ADD.B     @R13,0(R12)     ; Add LSDs ADC.B     1(R12)           ; Add carry to MSD

<b>ADD[.W]</b>	Add source to destination
<b>ADD.B</b>	Add source to destination
<b>Syntax</b>	ADD     src,dst     or     ADD.W   src,dst ADD.B   src,dst
<b>Operation</b>	src + dst -> dst
<b>Description</b>	The source operand is added to the destination operand. The source operand is not affected. The previous contents of the destination are lost.
<b>Status Bits</b>	N: Set if result is negative, reset if positive Z: Set if result is zero, reset otherwise C: Set if there is a carry from the result, cleared if not V: Set if an arithmetic overflow occurs, otherwise reset
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	R5 is increased by 10. The jump to TONI is performed on a carry.  ADD       #10,R5 JC        TONI               ; Carry occurred .....                       ; No carry
<b>Example</b>	R5 is increased by 10. The jump to TONI is performed on a carry.  ADD.B     #10,R5             ; Add 10 to Lowbyte of R5 JC        TONI               ; Carry occurred, if (R5) ≥ 246 [0Ah+0F6h] .....                       ; No carry



<b>ADDC[.W] ADDC.B</b>	Add source and carry to destination Add source and carry to destination
<b>Syntax</b>	ADDC            src,dst    or    ADDC.W    src,dst ADDC.B           src,dst
<b>Operation</b>	src + dst + C → dst
<b>Description</b>	The source operand and the carry bit (C) are added to the destination operand. The source operand is not affected. The previous contents of the destination are lost.
<b>Status Bits</b>	N: Set if result is negative, reset if positive Z: Set if result is zero, reset otherwise C: Set if there is a carry from the MSB of the result, reset otherwise V: Set if an arithmetic overflow occurs, otherwise reset
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	The 32-bit counter pointed to by R13 is added to a 32-bit counter, eleven words (20/2 + 2/2) above the pointer in R13.  ADD            @R13+,20(R13) ; ADD LSDs with no carry in ADDC            @R13+,20(R13) ; ADD MSDs with carry ...                                  ; resulting from the LSDs
<b>Example</b>	The 24-bit counter pointed to by R13 is added to a 24-bit counter, eleven words above the pointer in R13.  ADD.B         @R13+,10(R13) ; ADD LSDs with no carry in ADDC.B        @R13+,10(R13) ; ADD medium Bits with carry ADDC.B        @R13+,10(R13) ; ADD MSDs with carry ...                                  ; resulting from the LSDs

<b>AND[.W]</b>	Source AND destination
<b>AND.B</b>	Source AND destination
<b>Syntax</b>	AND        src,dst    or   AND.W src,dst AND.B     src,dst
<b>Operation</b>	src .AND. dst -> dst
<b>Description</b>	The source operand and the destination operand are logically ANDed. The result is placed into the destination.
<b>Status Bits</b>	N: Set if result MSB is set, reset if not set Z: Set if result is zero, reset otherwise C: Set if result is not zero, reset otherwise ( = .NOT. Zero) V: Reset

**Mode Bits**            OSCOFF, CPUOFF, and GIE are not affected.

**Example**             The bits set in R5 are used as a mask (#0AA55h) for the word addressed by TOM. If the result is zero, a branch is taken to label TONI.

```

MOV        #0AA55h,R5        ; Load mask into register R5
AND        R5,TOM            ; mask word addressed by TOM with R5
JZ         TONI              ;
.....                        ; Result is not zero
;
;
;
;            or
;
;
AND        #0AA55h,TOM
JZ         TONI

```

**Example**             The bits of mask #0A5h are logically ANDed with the low byte TOM. If the result is zero, a branch is taken to label TONI.

```

AND.B     #0A5h,TOM        ; mask Lowbyte TOM with 0A5h
JZ        TONI              ;
.....                        ; Result is not zero

```

<b>BIC[W]</b>	Clear bits in destination
<b>BIC.B</b>	Clear bits in destination
<b>Syntax</b>	BIC            src,dst    or   BIC.W src,dst BIC.B           src,dst
<b>Operation</b>	.NOT.src .AND. dst -> dst
<b>Description</b>	The inverted source operand and the destination operand are logically ANDed. The result is placed into the destination. The source operand is not affected.
<b>Status Bits</b>	Status bits are not affected.
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	The six MSBs of the RAM word LEO are cleared.  BIC            #0FC00h,LEO            ; Clear 6 MSBs in MEM(LEO)
<b>Example</b>	The five MSBs of the RAM byte LEO are cleared.  BIC.B           #0F8h,LEO            ; Clear 5 MSBs in Ram location LEO

<b>BIS[W]</b>	Set bits in destination
<b>BIS.B</b>	Set bits in destination
<b>Syntax</b>	BIS            src,dst    or   BIS.W        src,dst BIS.B           src,dst
<b>Operation</b>	src .OR. dst -> dst
<b>Description</b>	The source operand and the destination operand are logically ORed. The result is placed into the destination. The source operand is not affected.
<b>Status Bits</b>	Status bits are not affected.
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	The six LSBs of the RAM word TOM are set.  BIS            #003Fh,TOM; set the six LSBs in RAM location TOM
<b>Example</b>	The three MSBs of RAM byte TOM are set.  BIS.B        #0E0h,TOM        ; set the 3 MSBs in RAM location TOM

<b>BIT[.W]</b>	Test bits in destination
<b>BIT.B</b>	Test bits in destination
<b>Syntax</b>	BIT            src,dst    or BIT.W src,dst
<b>Operation</b>	src .AND. dst
<b>Description</b>	The source and destination operands are logically ANDed. The result affects only the status bits. The source and destination operands are not affected.
<b>Status Bits</b>	N: Set if MSB of result is set, reset otherwise Z: Set if result is zero, reset otherwise C: Set if result is not zero, reset otherwise (.NOT. Zero) V: Reset
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	If bit 9 of R8 is set, a branch is taken to label TOM.  <pre> BIT            #0200h,R8            ; bit 9 of R8 set? JNZ            TOM                 ; Yes, branch to TOM ...                                 ; No, proceed </pre>
<b>Example</b>	If bit 3 of R8 is set, a branch is taken to label TOM.  <pre> BIT.B          #8,R8 JC             TOM </pre>
<b>Example</b>	A serial communication receive bit (RCV) is tested. Because the carry bit is equal to the state of the tested bit while using the BIT instruction to test a single bit, the carry bit is used by the subsequent instruction; the read information is shifted into register RECBUF. <pre> ; ; Serial communication with LSB is shifted first: ; xxxx  xxxx  xxxx  xxxx BIT.B          #RCV,RCCTL         ; Bit info into carry RRC            RECBUF             ; Carry -&gt; MSB of RECBUF ; cxxx  cxxx .....                               ; repeat previous two instructions .....                               ; 8 times ; cccc  cccc ; ^       ^ ; MSB    LSB  ; Serial communication with MSB shifted first: BIT.B          #RCV,RCCTL         ; Bit info into carry RLC.B          RECBUF             ; Carry -&gt; LSB of RECBUF ; xxxx  xxxc .....                               ; repeat previous two instructions .....                               ; 8 times ; cccc  cccc ;         LSB ; MSB </pre>

<b>* BR, BRANCH</b>	Branch to ..... destination	
<b>Syntax</b>	BR	dst
<b>Operation</b>	dst → PC	
<b>Emulation</b>	MOV	dst,PC
<b>Description</b>	An unconditional branch is taken to an address anywhere in the 64K address space. All source addressing modes can be used. The branch instruction is a word instruction.	
<b>Status Bits</b>	Status bits are not affected.	
<b>Example</b>	Examples for all addressing modes are given.	
	BR	#EXEC ;Branch to label EXEC or direct branch (e.g. #0A4h) ; Core instruction MOV @PC+,PC
	BR	EXEC ; Branch to the address contained in EXEC ; Core instruction MOV X(PC),PC ; Indirect address
	BR	&EXEC ; Branch to the address contained in absolute ; address EXEC ; Core instruction MOV X(0),PC ; Indirect address
	BR	R5 ; Branch to the address contained in R5 ; Core instruction MOV R5,PC ; Indirect R5
	BR	@R5 ; Branch to the address contained in the word ; pointed to by R5. ; Core instruction MOV @R5,PC ; Indirect, indirect R5
	BR	@R5+ ; Branch to the address contained in the word pointed ; to by R5 and increment pointer in R5 afterwards. ; The next time—S/W flow uses R5 pointer—it can ; alter program execution due to access to ; next address in a table pointed to by R5 ; Core instruction MOV @R5,PC ; Indirect, indirect R5 with autoincrement
	BR	X(R5) ; Branch to the address contained in the address ; pointed to by R5 + X (e.g. table with address ; starting at X). X can be an address or a label ; Core instruction MOV X(R5),PC ; Indirect, indirect R5 + X

<b>CALL</b>	Subroutine		
<b>Syntax</b>	CALL	dst	
<b>Operation</b>	dst	-> tmp	dst is evaluated and stored
	SP - 2	-> SP	
	PC	-> @SP	PC updated to TOS
	tmp	-> PC	dst saved to PC
<b>Description</b>	A subroutine call is made to an address anywhere in the 64K address space. All addressing modes can be used. The return address (the address of the following instruction) is stored on the stack. The call instruction is a word instruction.		
<b>Status Bits</b>	Status bits are not affected.		
<b>Example</b>	Examples for all addressing modes are given.		
	CALL	#EXEC	; Call on label EXEC or immediate address (e.g. #0A4h) ; SP-2 → SP, PC+2 → @SP, @PC+ → PC
	CALL	EXEC	; Call on the address contained in EXEC ; SP-2 → SP, PC+2 → @SP, X(PC) → PC ; Indirect address
	CALL	&EXEC	; Call on the address contained in absolute address ; EXEC ; SP-2 → SP, PC+2 → @SP, X(0) → PC ; Indirect address
	CALL	R5	; Call on the address contained in R5 ; SP-2 → SP, PC+2 → @SP, R5 → PC ; Indirect R5
	CALL	@R5	; Call on the address contained in the word ; pointed to by R5 ; SP-2 → SP, PC+2 → @SP, @R5 → PC ; Indirect, indirect R5
	CALL	@R5+	; Call on the address contained in the word ; pointed to by R5 and increment pointer in R5. ; The next time—S/W flow uses R5 pointer— ; it can alter the program execution due to ; access to next address in a table pointed to by R5 ; SP-2 → SP, PC+2 → @SP, @R5 → PC ; Indirect, indirect R5 with autoincrement
	CALL	X(R5)	; Call on the address contained in the address pointed ; to by R5 + X (e.g. table with address starting at X) ; X can be an address or a label ; SP-2 → SP, PC+2 → @SP, X(R5) → PC ; Indirect, indirect R5 + X

<b>* CLR[.W]</b>	Clear destination
<b>* CLR.B</b>	Clear destination
<b>Syntax</b>	CLR        dst    or CLR.W dst CLR.B     dst
<b>Operation</b>	0 -> dst
<b>Emulation</b>	MOV        #0,dst MOV.B     #0,dst
<b>Description</b>	The destination operand is cleared.
<b>Status Bits</b>	Status bits are not affected.
<b>Example</b>	RAM word TONI is cleared.  CLR        TONI        ; 0 -> TONI
<b>Example</b>	Register R5 is cleared.  CLR        R5
<b>Example</b>	RAM byte TONI is cleared.  CLR.B     TONI        ; 0 -> TONI



<b>* CLRC</b>	Clear carry bit
<b>Syntax</b>	CLRC
<b>Operation</b>	0 → C
<b>Emulation</b>	BIC #1,SR
<b>Description</b>	The carry bit (C) is cleared. The clear carry instruction is a word instruction.
<b>Status Bits</b>	N: Not affected Z: Not affected C: Cleared V: Not affected
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	The 16-bit decimal counter pointed to by R13 is added to a 32-bit counter pointed to by R12.  CLRC ; C=0: defines start DADD @R13,0(R12) ; add 16-bit counter to low word of 32-bit counter DADC 2(R12) ; add carry to high word of 32-bit counter

<b>* CLRN</b>	Clear negative bit
<b>Syntax</b>	CLR N
<b>Operation</b>	0 → N or (.NOT.src .AND. dst → dst)
<b>Emulation</b>	BIC #4,SR
<b>Description</b>	The constant 04h is inverted (0FFFBh) and is logically ANDed with the destination operand. The result is placed into the destination. The clear negative bit instruction is a word instruction.
<b>Status Bits</b>	N: Reset to 0 Z: Not affected C: Not affected V: Not affected
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	The Negative bit in the status register is cleared. This avoids special treatment with negative numbers of the subroutine called.
	CLR N
	CALL SUBR
	.....
	.....
SUBR	JN SUBRET ; If input is negative: do nothing and return
	.....
	.....
	.....
SUBRET	RET

---

<b>* CLRZ</b>	Clear zero bit
<b>Syntax</b>	CLRZ
<b>Operation</b>	0 → Z or (.NOT.src .AND. dst → dst)
<b>Emulation</b>	BIC #2,SR
<b>Description</b>	The constant 02h is inverted (0FFFDh) and logically ANDed with the destination operand. The result is placed into the destination. The clear zero bit instruction is a word instruction.
<b>Status Bits</b>	N: Not affected Z: Reset to 0 C: Not affected V: Not affected
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	The zero bit in the status register is cleared.  CLRZ

<b>CMP[.W]</b>	Compare source and destination
<b>CMP.B</b>	Compare source and destination
<b>Syntax</b>	CMP        src,dst    or    CMP.W    src,dst CMP.B       src,dst
<b>Operation</b>	dst + .NOT.src + 1 or (dst – src)
<b>Description</b>	The source operand is subtracted from the destination operand. This is accomplished by adding the 1s complement of the source operand plus 1. The two operands are not affected and the result is not stored; only the status bits are affected.
<b>Status Bits</b>	N: Set if result is negative, reset if positive (src >= dst) Z: Set if result is zero, reset otherwise (src = dst) C: Set if there is a carry from the MSB of the result, reset otherwise V: Set if an arithmetic overflow occurs, otherwise reset
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	R5 and R6 are compared. If they are equal, the program continues at the label EQUAL.  CMP        R5,R6        ; R5 = R6? JEQ        EQUAL       ; YES, JUMP
<b>Example</b>	Two RAM blocks are compared. If they are not equal, the program branches to the label ERROR.  MOV    #NUM,R5            ; number of words to be compared MOV    #BLOCK1,R6       ; BLOCK1 start address in R6 MOV    #BLOCK2,R7       ; BLOCK2 start address in R7 L\$1    CMP    @R6+,0(R7)       ; Are Words equal? R6 increments JNZ    ERROR            ; No, branch to ERROR INCD   R7               ; Increment R7 pointer DEC    R5               ; Are all words compared? JNZ    L\$1               ; No, another compare
<b>Example</b>	The RAM bytes addressed by EDE and TONI are compared. If they are equal, the program continues at the label EQUAL.  CMP.B EDE,TONI        ; MEM(EDE) = MEM(TONI)? JEQ    EQUAL         ; YES, JUMP

<b>* DADC[.W]</b>	Add carry decimally to destination
<b>* DADC.B</b>	Add carry decimally to destination
<b>Syntax</b>	DADC        dst    or    DADC.W    src,dst DADC.B     dst
<b>Operation</b>	dst + C → dst (decimally)
<b>Emulation</b>	DADD        #0,dst DADD.B     #0,dst
<b>Description</b>	The carry bit (C) is added decimally to the destination.
<b>Status Bits</b>	N: Set if MSB is 1 Z: Set if dst is 0, reset otherwise C: Set if destination increments from 9999 to 0000, reset otherwise Set if destination increments from 99 to 00, reset otherwise V: Undefined
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	The four-digit decimal number contained in R5 is added to an eight-digit decimal number pointed to by R8.  <pre> CLRC                                ; Reset carry                                      ; next instruction's start condition is defined DADD        R5,0(R8)               ; Add LSDs + C DADC        2(R8)                   ; Add carry to MSD </pre>
<b>Example</b>	The two-digit decimal number contained in R5 is added to a four-digit decimal number pointed to by R8.  <pre> CLRC                                ; Reset carry                                      ; next instruction's start condition is defined DADD.B     R5,0(R8)               ; Add LSDs + C DADC        1(R8)                   ; Add carry to MSDs </pre>

<b>DADD[.W]</b>	Source and carry added decimally to destination
<b>DADD.B</b>	Source and carry added decimally to destination
<b>Syntax</b>	DADD        src,dst     or DADD.W   src,dst DADD.B     src,dst
<b>Operation</b>	src + dst + C → dst (decimally)
<b>Description</b>	The source operand and the destination operand are treated as four binary coded decimals (BCD) with positive signs. The source operand and the carry bit (C) are added decimally to the destination operand. The source operand is not affected. The previous contents of the destination are lost. The result is not defined for non-BCD numbers.
<b>Status Bits</b>	N: Set if the MSB is 1, reset otherwise Z: Set if result is zero, reset otherwise C: Set if the result is greater than 9999 Set if the result is greater than 99 V: Undefined
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	The eight-digit BCD number contained in R5 and R6 is added decimally to an eight-digit BCD number contained in R3 and R4 (R6 and R4 contain the MSDs).  <pre>CLRC                             ; clear carry DADD        R5,R3               ; add LSDs DADD        R6,R4               ; add MSDs with carry JC            OVERFLOW ; if carry occurs go to error handling routine</pre>
<b>Example</b>	The two-digit decimal counter in the RAM byte CNT is incremented by one.  <pre>CLRC                             ; clear carry DADD.B     #1,CNT               ; increment decimal counter</pre> <p>or</p> <pre>SETC DADD.B     #0,CNT               ; ≡ DADC.B     CNT</pre>

<b>* DEC.W]</b>	Decrement destination
<b>* DEC.B</b>	Decrement destination
<b>Syntax</b>	DEC        dst    or    DEC.W    dst DEC.B     dst
<b>Operation</b>	dst - 1 -> dst
<b>Emulation</b>	SUB     #1,dst
<b>Emulation</b>	SUB.B   #1,dst
<b>Description</b>	The destination operand is decremented by one. The original contents are lost.
<b>Status Bits</b>	N: Set if result is negative, reset if positive Z: Set if dst contained 1, reset otherwise C: Reset if dst contained 0, set otherwise V: Set if an arithmetic overflow occurs, otherwise reset. Set if initial value of destination was 08000h, otherwise reset. Set if initial value of destination was 080h, otherwise reset.
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	R10 is decremented by 1  DEC        R10        ; Decrement R10

; Move a block of 255 bytes from memory location starting with EDE to memory location starting with  
; TONI. Tables should not overlap: start of destination address TONI must not be within the range EDE  
; to EDE+0FEh  
;

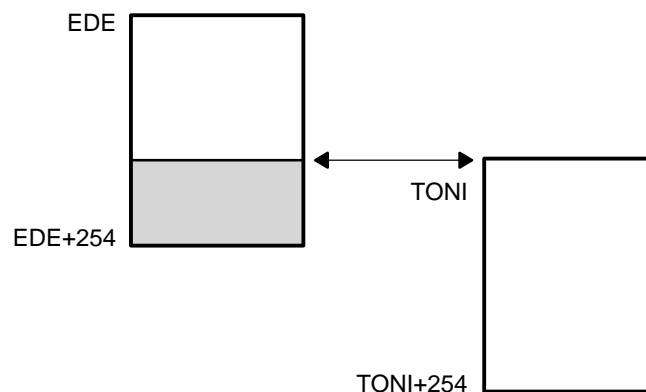
```

MOV        #EDE,R6
MOV        #255,R10
L$1        MOV.B     @R6+,TONI-EDE-1(R6)
DEC        R10
JNZ        L$1

```

; Do not transfer tables using the routine above with the overlap shown in Figure 3-12.

Figure 3-12. Decrement Overlap



<b>* DECD[.W]</b>	Double-decrement destination
<b>* DECD.B</b>	Double-decrement destination
<b>Syntax</b>	DECD       dst    or   DECD.W  dst DECD.B     dst
<b>Operation</b>	dst – 2 → dst
<b>Emulation</b>	SUB       #2,dst
<b>Emulation</b>	SUB.B     #2,dst
<b>Description</b>	The destination operand is decremented by two. The original contents are lost.
<b>Status Bits</b>	N: Set if result is negative, reset if positive Z: Set if dst contained 2, reset otherwise C: Reset if dst contained 0 or 1, set otherwise V: Set if an arithmetic overflow occurs, otherwise reset. Set if initial value of destination was 08001 or 08000h, otherwise reset. Set if initial value of destination was 081 or 080h, otherwise reset.
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	R10 is decremented by 2.

```

                                DECD       R10           ; Decrement R10 by two

```

```

; Move a block of 255 words from memory location starting with EDE to memory location
; starting with TONI
; Tables should not overlap: start of destination address TONI must not be within the
; range EDE to EDE+0FEh
;

```

```

                                MOV       #EDE,R6
                                MOV       #510,R10
L$1                               MOV       @R6+,TONI-EDE-2(R6)
                                DECD       R10
                                JNZ       L$1

```

**Example**                       Memory at location LEO is decremented by two.

```

                                DECD.B     LEO           ; Decrement MEM(LEO)

```

Decrement status byte STATUS by two.

```

                                DECD.B     STATUS

```



<b>* DINT</b>	Disable (general) interrupts
<b>Syntax</b>	DINT
<b>Operation</b>	0 → GIE or (0FFF7h .AND. SR → SR / .NOT.src .AND. dst → dst)
<b>Emulation</b>	BIC #8,SR
<b>Description</b>	All interrupts are disabled. The constant 08h is inverted and logically ANDed with the status register (SR). The result is placed into the SR.
<b>Status Bits</b>	Status bits are not affected.
<b>Mode Bits</b>	GIE is reset. OSCOFF and CPUOFF are not affected.
<b>Example</b>	The general interrupt enable (GIE) bit in the status register is cleared to allow a nondisrupted move of a 32-bit counter. This ensures that the counter is not modified during the move by any interrupt.  <pre>DINT           ; All interrupt events using the GIE bit are disabled NOP MOV    COUNTHI,R5 ; Copy counter MOV    COUNTLO,R6 EINT           ; All interrupt events using the GIE bit are enabled</pre>

---

**Note: Disable Interrupt**

If any code sequence needs to be protected from interruption, the DINT should be executed at least one instruction before the beginning of the uninterruptible sequence, or should be followed by a NOP instruction.

---

<b>* EINT</b>	Enable (general) interrupts
<b>Syntax</b>	EINT
<b>Operation</b>	1 → GIE or (0008h .OR. SR → SR / .src .OR. dst → dst)
<b>Emulation</b>	BIS #8,SR
<b>Description</b>	All interrupts are enabled. The constant #08h and the status register SR are logically ORed. The result is placed into the SR.
<b>Status Bits</b>	Status bits are not affected.
<b>Mode Bits</b>	GIE is set. OSCOFF and CPUOFF are not affected.
<b>Example</b>	The general interrupt enable (GIE) bit in the status register is set.

```

; Interrupt routine of ports P1.2 to P1.7
; P1IN is the address of the register where all port bits are read. P1IFG is the address of
; the register where all interrupt events are latched.
;
      PUSH.B  &P1IN
      BIC.B   @SP,&P1IFG ; Reset only accepted flags
      EINT    ; Preset port 1 interrupt flags stored on stack
              ; other interrupts are allowed
      BIT    #Mask,@SP
      JEQ    MaskOK ; Flags are present identically to mask: jump
      .....
MaskOK  BIC    #Mask,@SP
      .....
      INCD   SP ; Housekeeping: inverse to PUSH instruction
              ; at the start of interrupt subroutine. Corrects
              ; the stack pointer.

      RETI

```

**Note: Enable Interrupt**

The instruction following the enable interrupt instruction (EINT) is always executed, even if an interrupt service request is pending when the interrupts are enable.

<b>* INC[.W]</b>	Increment destination
<b>* INC.B</b>	Increment destination
<b>Syntax</b>	INC           dst    or   INC.W dst INC.B           dst
<b>Operation</b>	dst + 1 → dst
<b>Emulation</b>	ADD    #1,dst
<b>Description</b>	The destination operand is incremented by one. The original contents are lost.
<b>Status Bits</b>	N: Set if result is negative, reset if positive Z: Set if dst contained 0FFFFh, reset otherwise Set if dst contained 0FFh, reset otherwise C: Set if dst contained 0FFFFh, reset otherwise Set if dst contained 0FFh, reset otherwise V: Set if dst contained 07FFFh, reset otherwise Set if dst contained 07Fh, reset otherwise
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	The status byte, STATUS, of a process is incremented. When it is equal to 11, a branch to OVFL is taken.
	INC.B    STATUS CMP.B    #11,STATUS JEQ      OVFL

<b>* INCD[.W]</b>	Double-increment destination
<b>* INCD.B</b>	Double-increment destination
<b>Syntax</b>	INCD        dst    or INCD.W    dst INCD.B        dst
<b>Operation</b>	dst + 2 -> dst
<b>Emulation</b>	ADD        #2,dst
<b>Emulation</b>	ADD.B      #2,dst
<b>Example</b>	The destination operand is incremented by two. The original contents are lost.
<b>Status Bits</b>	N: Set if result is negative, reset if positive Z: Set if dst contained 0FFFEh, reset otherwise Set if dst contained 0FEh, reset otherwise C: Set if dst contained 0FFFEh or 0FFFFh, reset otherwise Set if dst contained 0FEh or 0FFh, reset otherwise V: Set if dst contained 07FFEh or 07FFFh, reset otherwise Set if dst contained 07Eh or 07Fh, reset otherwise
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	The item on the top of the stack (TOS) is removed without using a register.  ..... PUSH        R5            ; R5 is the result of a calculation, which is stored ; in the system stack INCD        SP            ; Remove TOS by double-increment from stack ; Do not use INCD.B, SP is a word-aligned ; register  RET
<b>Example</b>	The byte on the top of the stack is incremented by two.  INCD.B      0(SP)        ; Byte on TOS is increment by two

<b>* INV[.W]</b>	Invert destination
<b>* INV.B</b>	Invert destination
<b>Syntax</b>	INV        dst INV.B     dst
<b>Operation</b>	.NOT.dst -> dst
<b>Emulation</b>	XOR       #0FFFFh,dst
<b>Emulation</b>	XOR.B    #0FFh,dst
<b>Description</b>	The destination operand is inverted. The original contents are lost.
<b>Status Bits</b>	N: Set if result is negative, reset if positive Z: Set if dst contained 0FFFFh, reset otherwise Set if dst contained 0FFh, reset otherwise C: Set if result is not zero, reset otherwise ( = .NOT. Zero) Set if result is not zero, reset otherwise ( = .NOT. Zero) V: Set if initial destination operand was negative, otherwise reset
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	Content of R5 is negated (twos complement). MOV       #00AEh,R5 ;                                R5 = 000AEh INV       R5        ; Invert R5,                        R5 = 0FF51h INC       R5        ; R5 is now negated,                R5 = 0FF52h
<b>Example</b>	Content of memory byte LEO is negated. MOV.B    #0AEh,LEO ;                                MEM(LEO) = 0AEh INV.B    LEO        ; Invert LEO,                        MEM(LEO) = 051h INC.B    LEO        ; MEM(LEO) is negated, MEM(LEO) = 052h

<b>JC</b>	Jump if carry set
<b>JHS</b>	Jump if higher or same
<b>Syntax</b>	JC        label JHS        label
<b>Operation</b>	If C = 1: PC + 2 × offset → PC If C = 0: execute following instruction
<b>Description</b>	The status register carry bit (C) is tested. If it is set, the 10-bit signed offset contained in the instruction LSBs is added to the program counter. If C is reset, the next instruction following the jump is executed. JC (jump if carry/higher or same) is used for the comparison of unsigned numbers (0 to 65536).
<b>Status Bits</b>	Status bits are not affected.
<b>Example</b>	The P1IN.1 signal is used to define or control the program flow.  <pre> BIT        #01h,&amp;P1IN    ; State of signal → Carry JC        PROGA        ; If carry=1 then execute program routine A .....                   ; Carry=0, execute program here </pre>
<b>Example</b>	R5 is compared to 15. If the content is higher or the same, branch to LABEL.  <pre> CMP        #15,R5 JHS        LABEL        ; Jump is taken if R5 ≥ 15 .....                   ; Continue here if R5 &lt; 15 </pre>

<b>JEQ, JZ</b>	Jump if equal, jump if zero
<b>Syntax</b>	JEQ      label,    JZ      label
<b>Operation</b>	If Z = 1: PC + 2 × offset → PC If Z = 0: execute following instruction
<b>Description</b>	The status register zero bit (Z) is tested. If it is set, the 10-bit signed offset contained in the instruction LSBs is added to the program counter. If Z is not set, the instruction following the jump is executed.
<b>Status Bits</b>	Status bits are not affected.
<b>Example</b>	Jump to address TONI if R7 contains zero.  <pre>TST      R7                    ; Test R7 JZ        TONI                 ; if zero: JUMP</pre>
<b>Example</b>	Jump to address LEO if R6 is equal to the table contents.  <pre>CMP      R6,Table(R5)        ; Compare content of R6 with content of                                  ; MEM (table address + content of R5) JEQ      LEO                   ; Jump if both data are equal .....                         ; No, data are not equal, continue here</pre>
<b>Example</b>	Branch to LABEL if R5 is 0.  <pre>TST      R5 JZ        LABEL .....</pre>

<b>JGE</b>	Jump if greater or equal
<b>Syntax</b>	JGE      label
<b>Operation</b>	If (N .XOR. V) = 0 then jump to label: PC + 2 × offset → PC If (N .XOR. V) = 1 then execute the following instruction
<b>Description</b>	The status register negative bit (N) and overflow bit (V) are tested. If both N and V are set or reset, the 10-bit signed offset contained in the instruction LSBs is added to the program counter. If only one is set, the instruction following the jump is executed.  This allows comparison of signed integers.
<b>Status Bits</b>	Status bits are not affected.
<b>Example</b>	When the content of R6 is greater or equal to the memory pointed to by R7, the program continues at label EDE.  <pre> CMP      @R7,R6      ; R6 ≥ (R7)?, compare on signed numbers JGE      EDE          ; Yes, R6 ≥ (R7) .....                ; No, proceed ..... ..... </pre>



<b>JL</b>	Jump if less
<b>Syntax</b>	JL        label
<b>Operation</b>	If (N .XOR. V) = 1 then jump to label: PC + 2 × offset → PC If (N .XOR. V) = 0 then execute following instruction
<b>Description</b>	The status register negative bit (N) and overflow bit (V) are tested. If only one is set, the 10-bit signed offset contained in the instruction LSBs is added to the program counter. If both N and V are set or reset, the instruction following the jump is executed.  This allows comparison of signed integers.
<b>Status Bits</b>	Status bits are not affected.
<b>Example</b>	When the content of R6 is less than the memory pointed to by R7, the program continues at label EDE.  <pre> CMP     @R7,R6        ; R6 &lt; (R7)?, compare on signed numbers JL      EDE           ; Yes, R6 &lt; (R7) .....                ; No, proceed ..... ..... </pre>

<b>JMP</b>	Jump unconditionally
<b>Syntax</b>	JMP     label
<b>Operation</b>	$PC + 2 \times \text{offset} \rightarrow PC$
<b>Description</b>	The 10-bit signed offset contained in the instruction LSBs is added to the program counter.
<b>Status Bits</b>	Status bits are not affected.
<b>Hint:</b>	This one-word instruction replaces the BRANCH instruction in the range of -511 to +512 words relative to the current program counter.

<b>JN</b>	Jump if negative
<b>Syntax</b>	JN        label
<b>Operation</b>	if N = 1: PC + 2 × offset → PC if N = 0: execute following instruction
<b>Description</b>	The negative bit (N) of the status register is tested. If it is set, the 10-bit signed offset contained in the instruction LSBs is added to the program counter. If N is reset, the next instruction following the jump is executed.
<b>Status Bits</b>	Status bits are not affected.
<b>Example</b>	The result of a computation in R5 is to be subtracted from COUNT. If the result is negative, COUNT is to be cleared and the program continues execution in another path.
	<pre> SUB    R5,COUNT    ; COUNT - R5 -&gt; COUNT JN     L\$1         ; If negative continue with COUNT=0 at PC=L\$1 .....           ; Continue with COUNT≥0 ..... ..... ..... ..... CLR    COUNT ..... ..... ..... </pre>
L\$1	

<b>JNC</b>	Jump if carry not set
<b>JLO</b>	Jump if lower
<b>Syntax</b>	JNC     label JLO     label
<b>Operation</b>	if C = 0: PC + 2 × offset → PC if C = 1: execute following instruction
<b>Description</b>	The status register carry bit (C) is tested. If it is reset, the 10-bit signed offset contained in the instruction LSBs is added to the program counter. If C is set, the next instruction following the jump is executed. JNC (jump if no carry/lower) is used for the comparison of unsigned numbers (0 to 65536).
<b>Status Bits</b>	Status bits are not affected.
<b>Example</b>	The result in R6 is added in BUFFER. If an overflow occurs, an error handling routine at address ERROR is used.
	ADD     R6,BUFFER     ; BUFFER + R6 → BUFFER
	JNC     CONT           ; No carry, jump to CONT
ERROR	.....                 ; Error handler start
	.....
	.....
	.....
CONT	.....                   ; Continue with normal program flow
	.....
	.....
<b>Example</b>	Branch to STL2 if byte STATUS contains 1 or 0.
	CMP.B   #2,STATUS
	JLO     STL2           ; STATUS < 2
	.....                 ; STATUS ≥ 2, continue here

<b>JNE</b>	Jump if not equal	
<b>JNZ</b>	Jump if not zero	
<b>Syntax</b>	JNE	label
	JNZ	label
<b>Operation</b>	If Z = 0: PC + 2 × offset → PC If Z = 1: execute following instruction	
<b>Description</b>	The status register zero bit (Z) is tested. If it is reset, the 10-bit signed offset contained in the instruction LSBs is added to the program counter. If Z is set, the next instruction following the jump is executed.	
<b>Status Bits</b>	Status bits are not affected.	
<b>Example</b>	Jump to address TONI if R7 and R8 have different contents.	
	CMP	R7,R8 ; COMPARE R7 WITH R8
	JNE	TONI ; if different: jump
	.....	; if equal, continue

<b>MOV[.W]</b>	Move source to destination
<b>MOV.B</b>	Move source to destination
<b>Syntax</b>	MOV src,dst or MOV.W src,dst MOV.B src,dst
<b>Operation</b>	src -> dst
<b>Description</b>	The source operand is moved to the destination. The source operand is not affected. The previous contents of the destination are lost.
<b>Status Bits</b>	Status bits are not affected.
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	The contents of table EDE (word data) are copied to table TOM. The length of the tables must be 020h locations.
Loop	<pre> MOV #EDE,R10 ; Prepare pointer MOV #020h,R9 ; Prepare counter MOV @R10+,TOM-EDE-2(R10) ; Use pointer in R10 for both tables DEC R9 ; Decrement counter JNZ Loop ; Counter ≠ 0, continue copying ..... ; Copying completed ..... ..... </pre>
<b>Example</b>	The contents of table EDE (byte data) are copied to table TOM. The length of the tables should be 020h locations
Loop	<pre> MOV #EDE,R10 ; Prepare pointer MOV #020h,R9 ; Prepare counter MOV.B @R10+,TOM-EDE-1(R10) ; Use pointer in R10 for ; both tables DEC R9 ; Decrement counter JNZ Loop ; Counter ≠ 0, continue ; copying ..... ; Copying completed ..... ..... </pre>

<b>* NOP</b>	No operation
<b>Syntax</b>	NOP
<b>Operation</b>	None
<b>Emulation</b>	MOV #0, R3
<b>Description</b>	No operation is performed. The instruction may be used for the elimination of instructions during the software check or for defined waiting times.
<b>Status Bits</b>	Status bits are not affected.

The NOP instruction is mainly used for two purposes:

- To fill one, two, or three memory words
- To adjust software timing

---

**Note: Emulating No-Operation Instruction**

Other instructions can emulate the NOP function while providing different numbers of instruction cycles and code words. Some examples are:

Examples:

MOV	#0,R3	; 1 cycle, 1 word
MOV	0(R4),0(R4)	; 6 cycles, 3 words
MOV	@R4,0(R4)	; 5 cycles, 2 words
BIC	#0,EDE(R4)	; 4 cycles, 2 words
JMP	\$+2	; 2 cycles, 1 word
BIC	#0,R5	; 1 cycle, 1 word

However, care should be taken when using these examples to prevent unintended results. For example, if MOV 0(R4), 0(R4) is used and the value in R4 is 120h, then a security violation will occur with the watchdog timer (address 120h) because the security key was not used.

---

<b>* POP[.W]</b>	Pop word from stack to destination
<b>* POP.B</b>	Pop byte from stack to destination
<b>Syntax</b>	POP           dst POP.B        dst
<b>Operation</b>	@SP -> temp SP + 2 -> SP temp -> dst
<b>Emulation</b>	MOV           @SP+,dst    or   MOV.W    @SP+,dst
<b>Emulation</b>	MOV.B        @SP+,dst
<b>Description</b>	The stack location pointed to by the stack pointer (TOS) is moved to the destination. The stack pointer is incremented by two afterwards.
<b>Status Bits</b>	Status bits are not affected.
<b>Example</b>	The contents of R7 and the status register are restored from the stack.  POP           R7           ; Restore R7 POP           SR           ; Restore status register
<b>Example</b>	The contents of RAM byte LEO is restored from the stack.  POP.B        LEO           ; The low byte of the stack is moved to LEO.
<b>Example</b>	The contents of R7 is restored from the stack.  POP.B        R7            ; The low byte of the stack is moved to R7, ; the high byte of R7 is 00h
<b>Example</b>	The contents of the memory pointed to by R7 and the status register are restored from the stack.  POP.B        0(R7)        ; The low byte of the stack is moved to the ; the byte which is pointed to by R7 ; Example:   R7 = 203h ;                Mem(R7) = low byte of system stack ; Example:   R7 = 20Ah ;                Mem(R7) = low byte of system stack POP           SR           ; Last word on stack moved to the SR

**Note: The System Stack Pointer**

The system stack pointer (SP) is always incremented by two, independent of the byte suffix.



<b>PUSH[.W]</b>	Push word onto stack
<b>PUSH.B</b>	Push byte onto stack
<b>Syntax</b>	PUSH        src    or    PUSH.W    src PUSH.B       src
<b>Operation</b>	SP – 2 → SP src → @SP
<b>Description</b>	The stack pointer is decremented by two, then the source operand is moved to the RAM word addressed by the stack pointer (TOS).
<b>Status Bits</b>	Status bits are not affected.
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	The contents of the status register and R8 are saved on the stack.  PUSH        SR            ; save status register PUSH        R8            ; save R8
<b>Example</b>	The contents of the peripheral TCDAT is saved on the stack.  PUSH.B      &TCDAT       ; save data from 8-bit peripheral module, ; address TCDAT, onto stack

---

**Note: The System Stack Pointer**

The system stack pointer (SP) is always decremented by two, independent of the byte suffix.

---

<b>* RET</b>	Return from subroutine
<b>Syntax</b>	RET
<b>Operation</b>	@SP → PC SP + 2 → SP
<b>Emulation</b>	MOV @SP+, PC
<b>Description</b>	The return address pushed onto the stack by a CALL instruction is moved to the program counter. The program continues at the code address following the subroutine call.
<b>Status Bits</b>	Status bits are not affected.

**RETI** Return from interrupt

**Syntax** RETI

**Operation**  
 TOS → SR  
 SP + 2 → SP  
 TOS → PC  
 SP + 2 → SP

**Description** The status register is restored to the value at the beginning of the interrupt service routine by replacing the present SR contents with the TOS contents. The stack pointer (SP) is incremented by two.

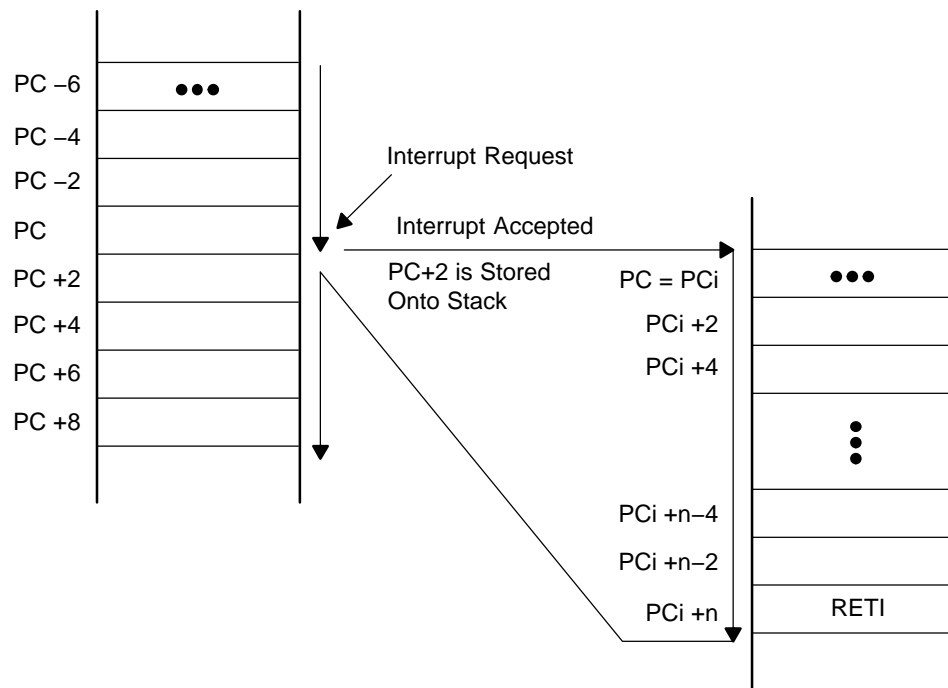
The program counter is restored to the value at the beginning of interrupt service. This is the consecutive step after the interrupted program flow. Restoration is performed by replacing the present PC contents with the TOS memory contents. The stack pointer (SP) is incremented.

**Status Bits**  
 N: restored from system stack  
 Z: restored from system stack  
 C: restored from system stack  
 V: restored from system stack

**Mode Bits** OSCOFF, CPUOFF, and GIE are restored from system stack.

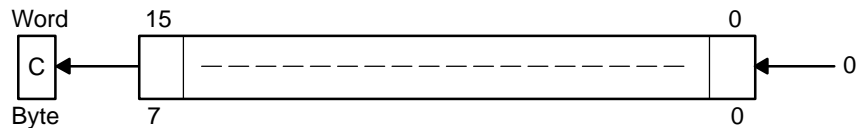
**Example** Figure 3–13 illustrates the main program interrupt.

Figure 3–13. Main Program Interrupt



<b>* RLA[.W]</b>	Rotate left arithmetically
<b>* RLA.B</b>	Rotate left arithmetically
<b>Syntax</b>	RLA     dst     or     RLA.W     dst RLA.B     dst
<b>Operation</b>	C ← MSB ← MSB-1 .... LSB+1 ← LSB ← 0
<b>Emulation</b>	ADD     dst,dst ADD.B   dst,dst
<b>Description</b>	The destination operand is shifted left one position as shown in Figure 3-14. The MSB is shifted into the carry bit (C) and the LSB is filled with 0. The RLA instruction acts as a signed multiplication by 2.  An overflow occurs if $dst \geq 04000h$ and $dst < 0C000h$ before operation is performed: the result has changed sign.

Figure 3-14. Destination Operand—Arithmetic Shift Left



An overflow occurs if  $dst \geq 040h$  and  $dst < 0C0h$  before the operation is performed: the result has changed sign.

<b>Status Bits</b>	N: Set if result is negative, reset if positive Z: Set if result is zero, reset otherwise C: Loaded from the MSB V: Set if an arithmetic overflow occurs: the initial value is $04000h \leq dst < 0C000h$ ; reset otherwise Set if an arithmetic overflow occurs: the initial value is $040h \leq dst < 0C0h$ ; reset otherwise
--------------------	---

**Mode Bits** OSCOFF, CPUOFF, and GIE are not affected.

**Example** R7 is multiplied by 2.

```
RLA        R7            ; Shift left R7 (× 2)
```

**Example** The low byte of R7 is multiplied by 4.

```
RLA.B     R7            ; Shift left low byte of R7 (× 2)
RLA.B     R7            ; Shift left low byte of R7 (× 4)
```

**Note: RLA Substitution**

The assembler does not recognize the instruction:

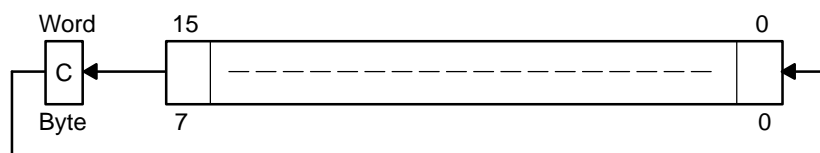
```
RLA        @R5+            nor     RLA.B     @R5+.
```

It must be substituted by:

```
ADD        @R5+,-2(R5)     or     ADD.B     @R5+,-1(R5).
```

<b>* RLC[.W]</b>	Rotate left through carry
<b>* RLC.B</b>	Rotate left through carry
<b>Syntax</b>	RLC     dst            or            RLC.W         dst RLC.B    dst
<b>Operation</b>	C ← MSB ← MSB-1 .... LSB+1 ← LSB ← C
<b>Emulation</b>	ADDC    dst,dst
<b>Description</b>	The destination operand is shifted left one position as shown in Figure 3-15. The carry bit (C) is shifted into the LSB and the MSB is shifted into the carry bit (C).

Figure 3-15. Destination Operand—Carry Left Shift



**Status Bits**

- N: Set if result is negative, reset if positive
- Z: Set if result is zero, reset otherwise
- C: Loaded from the MSB
- V: Set if an arithmetic overflow occurs  
the initial value is 04000h ≤ dst < 0C000h; reset otherwise
- Set if an arithmetic overflow occurs:  
the initial value is 040h ≤ dst < 0C0h; reset otherwise

**Mode Bits** OSCOFF, CPUOFF, and GIE are not affected.

**Example** R5 is shifted left one position.

```
RLC     R5            ; (R5 x 2) + C → R5
```

**Example** The input P1IN.1 information is shifted into the LSB of R5.

```
BIT.B    #2,&P1IN     ; Information → Carry  
RLC     R5            ; Carry=P0in.1 → LSB of R5
```

**Example** The MEM(LEO) content is shifted left one position.

```
RLC.B    LEO           ; Mem(LEO) x 2 + C → Mem(LEO)
```

**Note: RLC and RLC.B Substitution**

The assembler does not recognize the instruction:

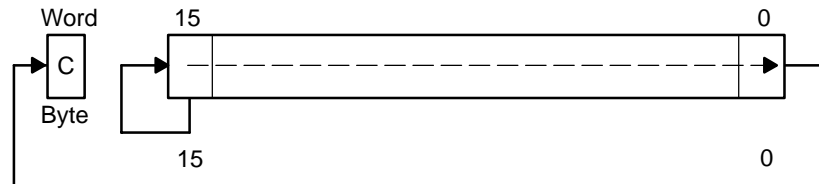
```
RLC     @R5+.
```

It must be substituted by:

```
ADDC    @R5+,-2(R5).
```

<b>RRA[.W]</b>	Rotate right arithmetically
<b>RRA.B</b>	Rotate right arithmetically
<b>Syntax</b>	RRA dst or RRA.W dst RRA.B dst
<b>Operation</b>	MSB → MSB, MSB → MSB-1, ... LSB+1 → LSB, LSB → C
<b>Description</b>	The destination operand is shifted right one position as shown in Figure 3-16. The MSB is shifted into the MSB, the MSB is shifted into the MSB-1, and the LSB+1 is shifted into the LSB.

Figure 3-16. Destination Operand—Arithmetic Right Shift



**Status Bits**

- N: Set if result is negative, reset if positive
- Z: Set if result is zero, reset otherwise
- C: Loaded from the LSB
- V: Reset

**Mode Bits** OSCOFF, CPUOFF, and GIE are not affected.

**Example** R5 is shifted right one position. The MSB retains the old value. It operates equal to an arithmetic division by 2.

```
RRA    R5        ; R5/2 → R5
```

; The value in R5 is multiplied by 0.75 (0.5 + 0.25).  
;

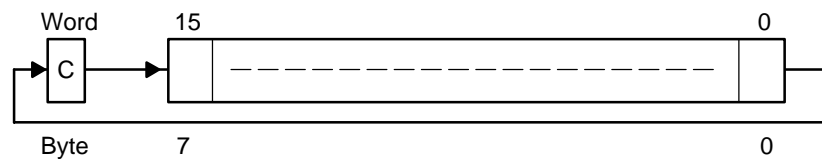
```
PUSH   R5        ; Hold R5 temporarily using stack
RRA    R5        ; R5 × 0.5 → R5
ADD    @SP+,R5   ; R5 × 0.5 + R5 = 1.5 × R5 → R5
RRA    R5        ; (1.5 × R5) × 0.5 = 0.75 × R5 → R5
.....
```

**Example** The low byte of R5 is shifted right one position. The MSB retains the old value. It operates equal to an arithmetic division by 2.

```
RRA.B  R5        ; R5/2 → R5: operation is on low byte only
                          ; High byte of R5 is reset
PUSH.B R5        ; R5 × 0.5 → TOS
RRA.B  @SP       ; TOS × 0.5 = 0.5 × R5 × 0.5 = 0.25 × R5 → TOS
ADD.B  @SP+,R5   ; R5 × 0.5 + R5 × 0.25 = 0.75 × R5 → R5
.....
```

<b>RRC[.W]</b>	Rotate right through carry
<b>RRC.B</b>	Rotate right through carry
<b>Syntax</b>	RRC dst or RRC.W dst RRC dst
<b>Operation</b>	C → MSB → MSB-1 ... LSB+1 → LSB → C
<b>Description</b>	The destination operand is shifted right one position as shown in Figure 3-17. The carry bit (C) is shifted into the MSB, the LSB is shifted into the carry bit (C).

Figure 3-17. Destination Operand—Carry Right Shift



<b>Status Bits</b>	N: Set if result is negative, reset if positive Z: Set if result is zero, reset otherwise C: Loaded from the LSB V: Set if initial destination is positive and initial carry is set, otherwise reset
--------------------	---

**Mode Bits** OSCOFF, CPUOFF, and GIE are not affected.

**Example** R5 is shifted right one position. The MSB is loaded with 1.

```
SETC          ; Prepare carry for MSB
RRC R5       ; R5/2 + 8000h → R5
```

**Example** R5 is shifted right one position. The MSB is loaded with 1.

```
SETC          ; Prepare carry for MSB
RRC.B R5     ; R5/2 + 80h → R5; low byte of R5 is used
```

<b>* SBC[.W]</b>	Subtract source and borrow/.NOT. carry from destination
<b>* SBC.B</b>	Subtract source and borrow/.NOT. carry from destination
<b>Syntax</b>	SBC     dst            or            SBC.W    dst SBC.B    dst
<b>Operation</b>	dst + 0FFFFh + C -> dst dst + 0FFh + C -> dst
<b>Emulation</b>	SUBC    #0,dst SUBC.B  #0,dst
<b>Description</b>	The carry bit (C) is added to the destination operand minus one. The previous contents of the destination are lost.
<b>Status Bits</b>	N: Set if result is negative, reset if positive Z: Set if result is zero, reset otherwise C: Set if there is a carry from the MSB of the result, reset otherwise. Set to 1 if no borrow, reset if borrow. V: Set if an arithmetic overflow occurs, reset otherwise.
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	The 16-bit counter pointed to by R13 is subtracted from a 32-bit counter pointed to by R12.  SUB            @R13,0(R12)                    ; Subtract LSDs SBC            2(R12)                            ; Subtract carry from MSD
<b>Example</b>	The 8-bit counter pointed to by R13 is subtracted from a 16-bit counter pointed to by R12.  SUB.B         @R13,0(R12)                    ; Subtract LSDs SBC.B         1(R12)                            ; Subtract carry from MSD

<b>Note: Borrow Implementation.</b>		
The borrow is treated as a .NOT. carry :	Borrow	Carry bit
	Yes	0
	No	1



<b>* SETC</b>	Set carry bit	
<b>Syntax</b>	SETC	
<b>Operation</b>	1 → C	
<b>Emulation</b>	BIS	#1,SR
<b>Description</b>	The carry bit (C) is set.	
<b>Status Bits</b>	N: Not affected Z: Not affected C: Set V: Not affected	
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.	
<b>Example</b>	Emulation of the decimal subtraction: Subtract R5 from R6 decimally Assume that R5 = 03987h and R6 = 04137h	
DSUB	ADD	#06666h,R5 ; Move content R5 from 0–9 to 6–0Fh ; R5 = 03987h + 06666h = 09FEDh
	INV	R5 ; Invert this (result back to 0–9) ; R5 = .NOT. R5 = 06012h
	SETC	; Prepare carry = 1
	DADD	R5,R6 ; Emulate subtraction by addition of: ; (010000h – R5 – 1) ; R6 = R6 + R5 + 1 ; R6 = 0150h

<b>* SETN</b>	Set negative bit
<b>Syntax</b>	SETN
<b>Operation</b>	1 → N
<b>Emulation</b>	BIS #4,SR
<b>Description</b>	The negative bit (N) is set.
<b>Status Bits</b>	N: Set Z: Not affected C: Not affected V: Not affected
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.

---

<b>* SETZ</b>	Set zero bit
<b>Syntax</b>	SETZ
<b>Operation</b>	1 → Z
<b>Emulation</b>	BIS #2,SR
<b>Description</b>	The zero bit (Z) is set.
<b>Status Bits</b>	N: Not affected Z: Set C: Not affected V: Not affected
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.

<b>SUB[.W]</b>	Subtract source from destination
<b>SUB.B</b>	Subtract source from destination
<b>Syntax</b>	SUB     src,dst     or     SUB.W     src,dst SUB.B     src,dst
<b>Operation</b>	dst + .NOT.src + 1 -> dst or [(dst - src -> dst)]
<b>Description</b>	The source operand is subtracted from the destination operand by adding the source operand's 1s complement and the constant 1. The source operand is not affected. The previous contents of the destination are lost.
<b>Status Bits</b>	N: Set if result is negative, reset if positive Z: Set if result is zero, reset otherwise C: Set if there is a carry from the MSB of the result, reset otherwise. Set to 1 if no borrow, reset if borrow. V: Set if an arithmetic overflow occurs, otherwise reset
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	See example at the SBC instruction.
<b>Example</b>	See example at the SBC.B instruction.

**Note: Borrow Is Treated as a .NOT.**

The borrow is treated as a .NOT. carry :	Borrow	Carry bit
	Yes	0
	No	1

<b>SUBC[.W]SBB[.W]</b>	Subtract source and borrow/.NOT. carry from destination
<b>SUBC.B,SBB.B</b>	Subtract source and borrow/.NOT. carry from destination
<b>Syntax</b>	SUBC    src,dst    or    SUBC.W    src,dst    or SBB    src,dst    or    SBB.W    src,dst SUBC.B src,dst    or    SBB.B    src,dst
<b>Operation</b>	dst + .NOT.src + C → dst or (dst – src – 1 + C → dst)
<b>Description</b>	The source operand is subtracted from the destination operand by adding the source operand's 1s complement and the carry bit (C). The source operand is not affected. The previous contents of the destination are lost.
<b>Status Bits</b>	N: Set if result is negative, reset if positive. Z: Set if result is zero, reset otherwise. C: Set if there is a carry from the MSB of the result, reset otherwise. Set to 1 if no borrow, reset if borrow. V: Set if an arithmetic overflow occurs, reset otherwise.
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	Two floating point mantissas (24 bits) are subtracted. LSBs are in R13 and R10, MSBs are in R12 and R9.  SUB.W    R13,R10    ; 16-bit part, LSBs SUBC.B   R12,R9    ; 8-bit part, MSBs
<b>Example</b>	The 16-bit counter pointed to by R13 is subtracted from a 16-bit counter in R10 and R11(MSD).  SUB.B    @R13+,R10            ; Subtract LSDs without carry SUBC.B   @R13,R11            ; Subtract MSDs with carry ...                                ; resulting from the LSDs

---

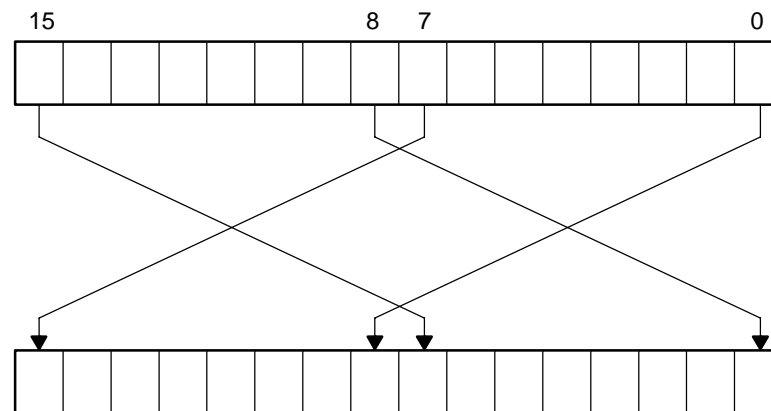
**Note: Borrow Implementation**

The borrow is treated as a .NOT. carry :	Borrow	Carry bit
	Yes	0
	No	1

---

<b>SWPB</b>	Swap bytes
<b>Syntax</b>	SWPB dst
<b>Operation</b>	Bits 15 to 8 <-> bits 7 to 0
<b>Description</b>	The destination operand high and low bytes are exchanged as shown in Figure 3–18.
<b>Status Bits</b>	Status bits are not affected.
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.

Figure 3–18. Destination Operand Byte Swap



**Example**

```
MOV    #040BFh,R7    ; 0100000010111111 -> R7
SWPB   R7             ; 1011111101000000 in R7
```

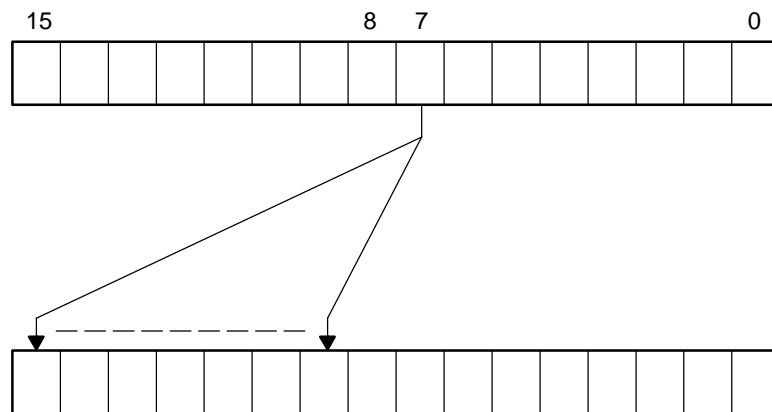
**Example**

The value in R5 is multiplied by 256. The result is stored in R5,R4.

```
SWPB   R5             ;
MOV    R5,R4         ;Copy the swapped value to R4
BIC    #0FF00h,R5    ;Correct the result
BIC    #00FFh,R4     ;Correct the result
```

<b>SXT</b>	Extend Sign
<b>Syntax</b>	SXT     dst
<b>Operation</b>	Bit 7 → Bit 8 ..... Bit 15
<b>Description</b>	The sign of the low byte is extended into the high byte as shown in Figure 3–19.
<b>Status Bits</b>	N: Set if result is negative, reset if positive Z: Set if result is zero, reset otherwise C: Set if result is not zero, reset otherwise (.NOT. Zero) V: Reset
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.

Figure 3–19. Destination Operand Sign Extension



**Example**                    R7 is loaded with the P1IN value. The operation of the sign-extend instruction expands bit 8 to bit 15 with the value of bit 7. R7 is then added to R6.

```
MOV.B  &P1IN,R7      ; P1IN = 080h:      .... 1000 0000
SXT    R7             ; R7 = 0FF80h:     1111 1111 1000 0000
```

<b>* TST[.W]</b>	Test destination
<b>* TST.B</b>	Test destination
<b>Syntax</b>	TST        dst    or TST.W dst TST.B       dst
<b>Operation</b>	dst + 0FFFFh + 1 dst + 0FFh + 1
<b>Emulation</b>	CMP        #0,dst CMP.B      #0,dst
<b>Description</b>	The destination operand is compared with zero. The status bits are set according to the result. The destination is not affected.
<b>Status Bits</b>	N: Set if destination is negative, reset if positive Z: Set if destination contains zero, reset otherwise C: Set V: Reset
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	R7 is tested. If it is negative, continue at R7NEG; if it is positive but not zero, continue at R7POS.
	<pre>                 TST    R7            ; Test R7                 JN     R7NEG        ; R7 is negative                 JZ     R7ZERO       ; R7 is zero R7POS        .....                ; R7 is positive but not zero R7NEG        .....                ; R7 is negative R7ZERO       .....                ; R7 is zero </pre>
<b>Example</b>	The low byte of R7 is tested. If it is negative, continue at R7NEG; if it is positive but not zero, continue at R7POS.
	<pre>                 TST.B  R7            ; Test low byte of R7                 JN     R7NEG        ; Low byte of R7 is negative                 JZ     R7ZERO       ; Low byte of R7 is zero R7POS        .....                ; Low byte of R7 is positive but not zero R7NEG        .....                ; Low byte of R7 is negative R7ZERO       .....                ; Low byte of R7 is zero </pre>



<b>XOR[.W]</b>	Exclusive OR of source with destination
<b>XOR.B</b>	Exclusive OR of source with destination
<b>Syntax</b>	<code>XOR src,dst or XOR.W src,dst</code> <code>XOR.B src,dst</code>
<b>Operation</b>	<code>src .XOR. dst -&gt; dst</code>
<b>Description</b>	The source and destination operands are exclusive ORed. The result is placed into the destination. The source operand is not affected.
<b>Status Bits</b>	N: Set if result MSB is set, reset if not set Z: Set if result is zero, reset otherwise C: Set if result is not zero, reset otherwise ( = .NOT. Zero) V: Set if both operands are negative
<b>Mode Bits</b>	OSCOFF, CPUOFF, and GIE are not affected.
<b>Example</b>	The bits set in R6 toggle the bits in the RAM word TONI. <code>XOR R6,TONI ; Toggle bits of word TONI on the bits set in R6</code>
<b>Example</b>	The bits set in R6 toggle the bits in the RAM byte TONI. <code>XOR.B R6,TONI ; Toggle bits of byte TONI on the bits set in</code> <code>; low byte of R6</code>
<b>Example</b>	Reset to 0 those bits in low byte of R7 that are different from bits in RAM byte EDE. <code>XOR.B EDE,R7 ; Set different bit to "1s"</code> <code>INV.B R7 ; Invert Lowbyte, Highbyte is 0h</code>

### 3.4.4 Instruction Cycles and Lengths

The number of CPU clock cycles required for an instruction depends on the instruction format and the addressing modes used - not the instruction itself. The number of clock cycles refers to the MCLK.

#### Interrupt and Reset Cycles

Table 3–14 lists the CPU cycles for interrupt overhead and reset.

Table 3–14. Interrupt and Reset Cycles

Action	No. of Cycles	Length of Instruction
Return from interrupt (RETI)	5	1
Interrupt accepted	6	–
WDT reset	4	–
Reset (RST/NMI)	4	–

#### Format-II (Single Operand) Instruction Cycles and Lengths

Table 3–15 lists the length and CPU cycles for all addressing modes of format-II instructions.

Table 3–15. Format-II Instruction Cycles and Lengths

Addressing Mode	No. of Cycles			Length of Instruction	Example
	RRA, RRC SWPB, SXT	PUSH	CALL		
Rn	1	3	4	1	SWPB R5
@Rn	3	4	4	1	RRC @R9
@Rn+	3	5	5	1	SWPB @R10+
#N	(See note)	4	5	2	CALL #0F00h
X(Rn)	4	5	5	2	CALL 2(R7)
EDE	4	5	5	2	PUSH EDE
&EDE	4	5	5	2	SXT &EDE

**Note: Instruction Format II Immediate Mode**

Do not use instructions RRA, RRC, SWPB, and SXT with the immediate mode in the destination field. Use of these in the immediate mode results in an unpredictable program operation.

#### Format-III (Jump) Instruction Cycles and Lengths

All jump instructions require one code word, and take two CPU cycles to execute, regardless of whether the jump is taken or not.

## Format-I (Double Operand) Instruction Cycles and Lengths

Table 3–16 lists the length and CPU cycles for all addressing modes of format-I instructions.

Table 3–16. Format I Instruction Cycles and Lengths

Addressing Mode		No. of Cycles	Length of Instruction		Example
Src	Dst				
Rn	Rm	1	1	MOV	R5, R8
	PC	2	1	BR	R9
	x(Rm)	4	2	ADD	R5, 4(R6)
	EDE	4	2	XOR	R8, EDE
	&EDE	4	2	MOV	R5, &EDE
@Rn	Rm	2	1	AND	@R4, R5
	PC	2	1	BR	@R8
	x(Rm)	5	2	XOR	@R5, 8(R6)
	EDE	5	2	MOV	@R5, EDE
	&EDE	5	2	XOR	@R5, &EDE
@Rn+	Rm	2	1	ADD	@R5+, R6
	PC	3	1	BR	@R9+
	x(Rm)	5	2	XOR	@R5, 8(R6)
	EDE	5	2	MOV	@R9+, EDE
	&EDE	5	2	MOV	@R9+, &EDE
#N	Rm	2	2	MOV	#20, R9
	PC	3	2	BR	#2AEh
	x(Rm)	5	3	MOV	#0300h, 0(SP)
	EDE	5	3	ADD	#33, EDE
	&EDE	5	3	ADD	#33, &EDE
x(Rn)	Rm	3	2	MOV	2(R5), R7
	PC	3	2	BR	2(R6)
	TONI	6	3	MOV	4(R7), TONI
	x(Rm)	6	3	ADD	4(R4), 6(R9)
	&TONI	6	3	MOV	2(R4), &TONI
EDE	Rm	3	2	AND	EDE, R6
	PC	3	2	BR	EDE
	TONI	6	3	CMP	EDE, TONI
	x(Rm)	6	3	MOV	EDE, 0(SP)
	&TONI	6	3	MOV	EDE, &TONI
&EDE	Rm	3	2	MOV	&EDE, R8
	PC	3	2	BRA	&EDE
	TONI	6	3	MOV	&EDE, TONI
	x(Rm)	6	3	MOV	&EDE, 0(SP)
	&TONI	6	3	MOV	&EDE, &TONI

### 3.4.5 Instruction Set Description

The instruction map is shown in Figure 3–20 and the complete instruction set is summarized in Table 3–17.

Figure 3–20. Core Instruction Map

	000	040	080	0C0	100	140	180	1C0	200	240	280	2C0	300	340	380	3C0
0xxx																
4xxx																
8xxx																
Cxxx																
1xxx	RRC	RRC.B	SWPB		RRA	RRA.B	SXT		PUSH	PUSH.B	CALL		RETI			
14xx																
18xx																
1Cxx																
20xx	JNE/JNZ															
24xx	JEQ/JZ															
28xx	JNC															
2Cxx	JC															
30xx	JN															
34xx	JGE															
38xx	JL															
3Cxx	JMP															
4xxx	MOV, MOV.B															
5xxx	ADD, ADD.B															
6xxx	ADDC, ADDC.B															
7xxx	SUBC, SUBC.B															
8xxx	SUB, SUB.B															
9xxx	CMP, CMP.B															
Axxx	DADD, DADD.B															
Bxxx	BIT, BIT.B															
Cxxx	BIC, BIC.B															
Dxxx	BIS, BIS.B															
Exxx	XOR, XOR.B															
Fxxx	AND, AND.B															

Table 3–17. MSP430 Instruction Set

Mnemonic		Description		V	N	Z	C
ADC(.B)†	dst	Add C to destination	dst + C → dst	*	*	*	*
ADD(.B)	src, dst	Add source to destination	src + dst → dst	*	*	*	*
ADDC(.B)	src, dst	Add source and C to destination	src + dst + C → dst	*	*	*	*
AND(.B)	src, dst	AND source and destination	src .and. dst → dst	0	*	*	*
BIC(.B)	src, dst	Clear bits in destination	.not.src .and. dst → dst	–	–	–	–
BIS(.B)	src, dst	Set bits in destination	src .or. dst → dst	–	–	–	–
BIT(.B)	src, dst	Test bits in destination	src .and. dst	0	*	*	*
BR†	dst	Branch to destination	dst → PC	–	–	–	–
CALL	dst	Call destination	PC+2 → stack, dst → PC	–	–	–	–
CLR(.B)†	dst	Clear destination	0 → dst	–	–	–	–
CLRC†		Clear C	0 → C	–	–	–	0
CLRN†		Clear N	0 → N	–	0	–	–
CLRZ†		Clear Z	0 → Z	–	–	0	–
CMP(.B)	src, dst	Compare source and destination	dst – src	*	*	*	*
DADC(.B)†	dst	Add C decimally to destination	dst + C → dst (decimally)	*	*	*	*
DADD(.B)	src, dst	Add source and C decimally to dst.	src + dst + C → dst (decimally)	*	*	*	*
DEC(.B)†	dst	Decrement destination	dst – 1 → dst	*	*	*	*
DECD(.B)†	dst	Double-decrement destination	dst – 2 → dst	*	*	*	*
DINT†		Disable interrupts	0 → GIE	–	–	–	–
EINT†		Enable interrupts	1 → GIE	–	–	–	–
INC(.B)†	dst	Increment destination	dst + 1 → dst	*	*	*	*
INCD(.B)†	dst	Double-increment destination	dst + 2 → dst	*	*	*	*
INV(.B)†	dst	Invert destination	.not.dst → dst	*	*	*	*
JC/JHS	label	Jump if C set/Jump if higher or same		–	–	–	–
JEQ/JZ	label	Jump if equal/Jump if Z set		–	–	–	–
JGE	label	Jump if greater or equal		–	–	–	–
JL	label	Jump if less		–	–	–	–
JMP	label	Jump	PC + 2 x offset → PC	–	–	–	–
JN	label	Jump if N set		–	–	–	–
JNC/JLO	label	Jump if C not set/Jump if lower		–	–	–	–
JNE/JNZ	label	Jump if not equal/Jump if Z not set		–	–	–	–
MOV(.B)	src, dst	Move source to destination	src → dst	–	–	–	–
NOPT		No operation		–	–	–	–
POP(.B)†	dst	Pop item from stack to destination	@SP → dst, SP+2 → SP	–	–	–	–
PUSH(.B)	src	Push source onto stack	SP – 2 → SP, src → @SP	–	–	–	–
RET†		Return from subroutine	@SP → PC, SP + 2 → SP	–	–	–	–
RETI		Return from interrupt		*	*	*	*
RLA(.B)†	dst	Rotate left arithmetically		*	*	*	*
RLC(.B)†	dst	Rotate left through C		*	*	*	*
RRA(.B)	dst	Rotate right arithmetically		0	*	*	*
RRC(.B)	dst	Rotate right through C		*	*	*	*
SBC(.B)†	dst	Subtract not(C) from destination	dst + 0FFFFh + C → dst	*	*	*	*
SETC†		Set C	1 → C	–	–	–	1
SETN†		Set N	1 → N	–	1	–	–
SETZ†		Set Z	1 → C	–	–	1	–
SUB(.B)	src, dst	Subtract source from destination	dst + .not.src + 1 → dst	*	*	*	*
SUBC(.B)	src, dst	Subtract source and not(C) from dst.	dst + .not.src + C → dst	*	*	*	*
SWPB	dst	Swap bytes		–	–	–	–
SXT	dst	Extend sign		0	*	*	*
TST(.B)†	dst	Test destination	dst + 0FFFFh + 1	0	*	*	1
XOR(.B)	src, dst	Exclusive OR source and destination	src .xor. dst → dst	*	*	*	*

† Emulated Instruction

# FLL+ Clock Module

---

---

---

---

The FLL+ clock module provides the clocks for MSP430x4xx devices. This chapter discusses the FLL+ clock module. The FLL+ clock module is implemented in all MSP430x4xx devices.

<b>Topic</b>	<b>Page</b>
4.1 FLL+ Clock Module Introduction .....	4-2
4.2 FLL+ Clock Module Operation .....	4-5
4.3 FLL+ Clock Module Registers .....	4-11

---

## 4.1 FLL+ Clock Module Introduction

The frequency-locked loop (FLL+) clock module supports low system cost and ultralow-power consumption. Using three internal clock signals, the user can select the best balance of performance and low power consumption. The FLL+ features digital frequency-locked loop (FLL) hardware. The FLL operates together with a digital modulator and stabilizes the internal digitally controlled oscillator (DCO) frequency to a programmable multiple of the LFXT1 watch crystal frequency. The FLL+ clock module can be configured to operate without any external components, with one or two external crystals, or with resonators, under full software control.

The FLL+ clock module includes two or three clock sources:

- LFXT1CLK: Low-frequency/high-frequency oscillator that can be used either with low-frequency 32768-Hz watch crystals, or standard crystals or resonators in the 450-kHz to 8-MHz range.
- XT2CLK: Optional high-frequency oscillator that can be used with standard crystals, resonators, or external clock sources in the 450-kHz to 8-MHz range.
- DCOCLK: Internal digitally controlled oscillator (DCO) with RC-type characteristics, stabilized by the FLL.
- Four clock signals are available from the FLL+ module:
- ACLK: Auxiliary clock. The ACLK is the LFXT1CLK clock source. ACLK is software selectable for individual peripheral modules.
- ACLK/n: Buffered output of the ACLK. The ACLK/n is ACLK divided by 1,2,4 or 8 and only used externally.
- MCLK: Master clock. MCLK is software selectable as LFXT1CLK, XT2CLK (if available), or DCOCLK. MCLK can be divided by 1, 2, 4, or 8 within the FLL block. MCLK is used by the CPU and system.
- SMCLK: Sub-main clock. SMCLK is software selectable as XT2CLK (if available), or DCOCLK. SMCLK is software selectable for individual peripheral modules.

The block diagram of the FLL+ clock module is shown in Figure 4–1 for the MSP430x44x and MSP430x43x. The block diagram of the FLL+ clock module is shown in Figure 4–2 for the MSP430x42x and MSP430x41x.

Figure 4–1. MSP430x44x and MSP430x43x Frequency-Locked Loop

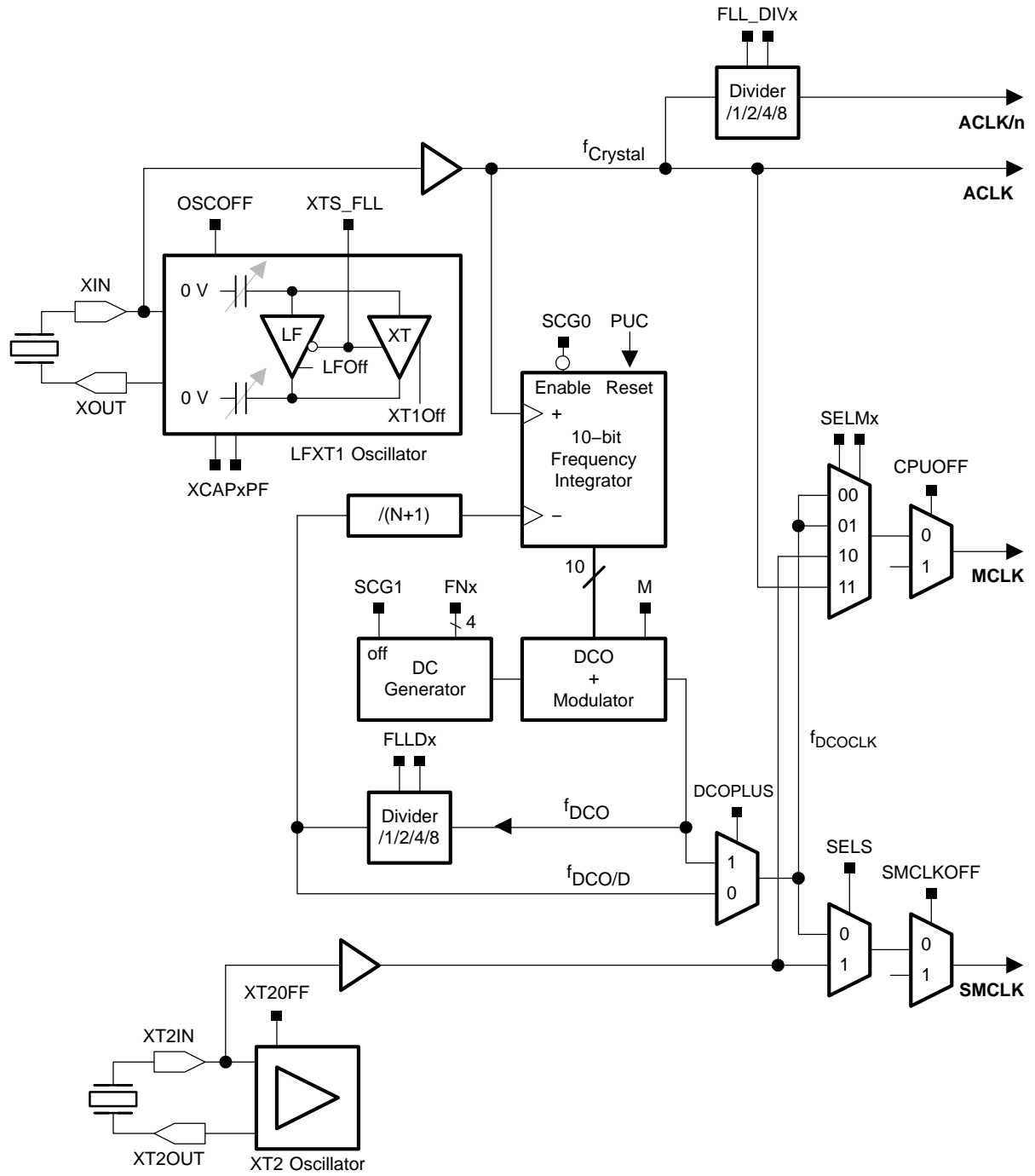
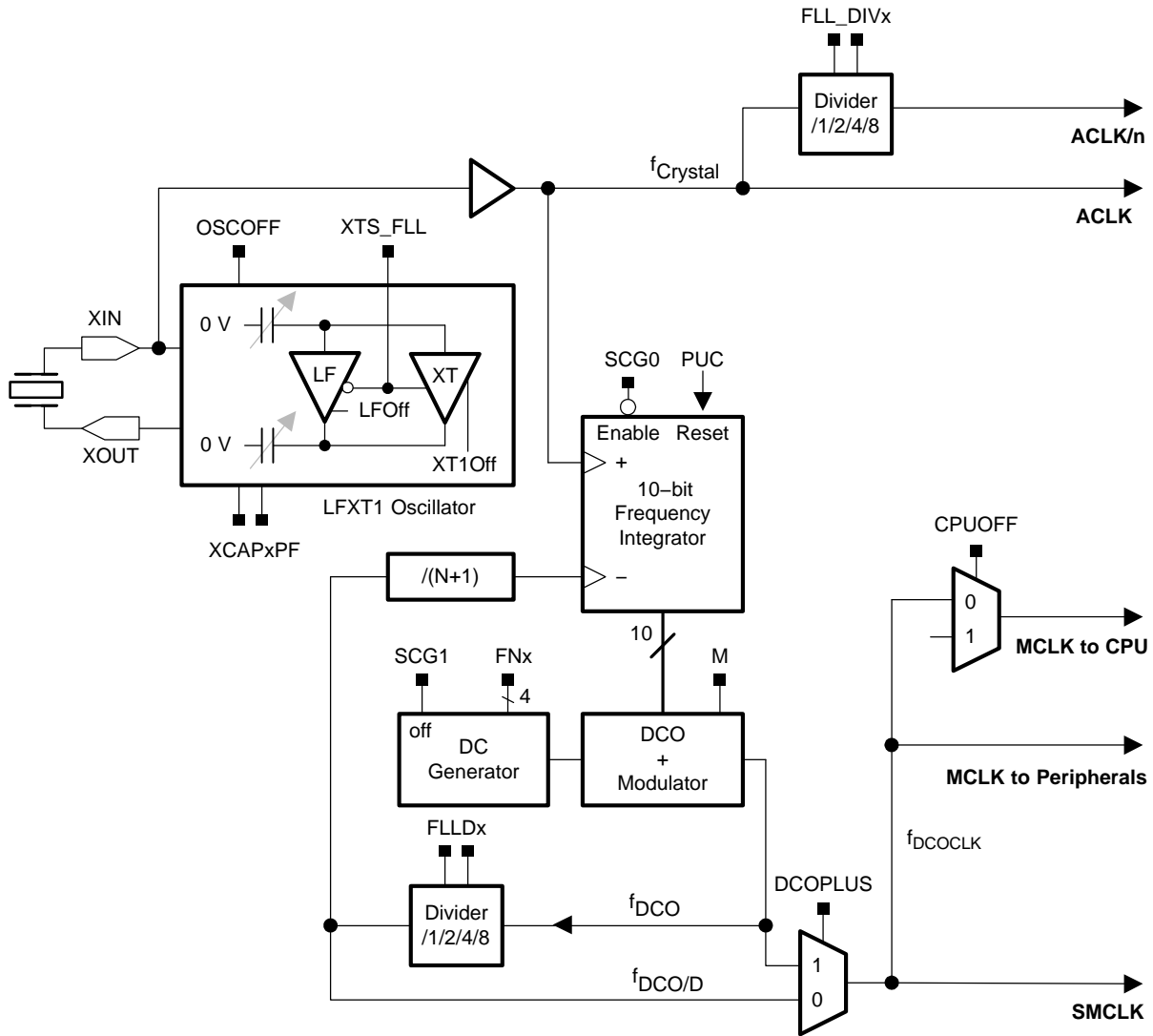




Figure 4–2. MSP430x42x and MSP430x41x Frequency-Locked Loop



## 4.2 FLL+ Clock Module Operation

After a PUC, MCLK and SMCLK are sourced from DCOCLK at 32 times the ACLK frequency. When a 32,768-Hz crystal is used for ACLK, MCLK and SMCLK will stabilize to 1.048576 MHz.

Status register control bits SCG0, SCG1, OSCOFF, and CPUOFF configure the MSP430 operating modes and enable or disable components of the FLL+ clock module. See Chapter *System Resets, Interrupts and Operating Modes*. The SCFQCTL, SCFIO, SCF11, FLL\_CTL0, and FLL\_CTL1 registers configure the FLL+ clock module. The FLL+ can be configured or reconfigured by software at any time during program execution.

Example,  $MCLK = 64 \times ACLK = 2097152$

```

BIC    #GIE,SR           ; Disable interrupts
MOV.B  #(64-1),&SCFQCTL  ; MCLK = 64 * ACLK, DCOPLUS=0
MOV.B  #FN_2,&SCFIO      ; Select DCO range
BIS    #GIE,SR           ; Enable interrupts

```

### 4.2.1 FLL+ Clock features for Low-Power Applications

Conflicting requirements typically exist in battery powered MSP430x4xx applications:

- Low clock frequency for energy conservation and time keeping
- High clock frequency for fast reaction to events and fast burst processing capability
- Clock stability over operating temperature and supply voltage

The FLL+ clock module addresses the above conflicting requirements by allowing the user to select from the three available clock signals: ACLK, MCLK, and SMCLK. For optimal low-power performance, the ACLK can be configured to oscillate with a low-power 32,786-Hz watch-crystal, providing a stable time base for the system and low power stand-by operation. The MCLK can be configured to operate from the on-chip DCO, stabilized by the FLL, and can activate when requested by interrupt events.

The digital frequency-locked loop provides decreased start-time and stabilization delay over an analog phase-locked loop. A phase-locked loop takes hundreds or thousands of clock cycles to start and stabilize. The FLL starts immediately at its previous setting.

### 4.2.2 LFXT1 Oscillator

The LFXT1 oscillator supports ultralow-current consumption using a 32,768-Hz watch crystal in LF mode (XTS\_FLL = 0). A watch crystal connects to XIN and XOUT without any external components.

The LFXT1 oscillator supports high-speed crystals or resonators when in HF mode (XTS\_FLL = 1). The high-speed crystal or resonator connects to XIN and XOUT.

LFXT1 may be used with an external clock signal on the XIN pin when XTS\_FLL = 1. The input frequency range is ~1 Hz – 8 MHz. When the input frequency is below 450 KHz, the XT1OF bit may be set preventing the CPU from being clocked from the external frequency.

The software-selectable XCAPxPF bits configure the internally provided load capacitance for the LFXT1 crystal. The internal pin capacitance plus the parasitic 2 pF pin capacitance combine serially to form the load capacitance. The load capacitance can be selected as 1, 6, 8, or 10 pF. Additional external capacitors can be added if necessary.

Software can disable LFXT1 by setting OSCOFF if this signal does not source MCLK (SELM ≠ 3 or CPUOFF = 1 ).

---

**Note: LFXT1 Oscillator Characteristics**

Low-frequency crystals often require hundreds of milliseconds to start up, depending on the crystal.

Ultralow-power oscillators such as the LFXT1 in LF mode should be guarded from noise coupling from other sources. The crystal should be placed as close as possible to the MSP430 with the crystal housing grounded and the crystal traces guarded with ground traces.

The default value of XCAPxPF is 0, providing a crystal load capacitance of ~1 pF. Reliable crystal operation may not be achieved unless the crystal is provided with the proper load capacitance—either by selection of XCAPxPF values or by external capacitors.

---

### 4.2.3 XT2 Oscillator

Some devices have a second crystal oscillator, XT2. XT2 sources XT2CLK and its characteristics are identical to LFXT1 in HF mode, except XT2 does not have internal load capacitors. The required load capacitance for the high frequency crystal or resonator must be provided externally.

The XT2OFF bit disables the XT2 oscillator if XT2CLK is unused for MCLK (SELMx ≠ 2 or CPUOFF = 1) and SMCLK (SELS = 0 or SMCLKOFF = 1).

XT2 may be used with external clock signals on the XT2IN pin. When used with an external signal, the external frequency must meet the datasheet parameters for XT2.

#### 4.2.4 Digitally-Controlled Oscillator (DCO)

The DCO is an integrated ring oscillator with RC-type characteristics. The DCO frequency is stabilized by the FLL to a multiple of ACLK as defined by N, the lowest 7 bits of the SCFQCTL register.

The DCOPLUS bit sets the  $f_{\text{DCOCLK}}$  frequency to  $f_{\text{DCO}}$  or  $f_{\text{DCO}/D}$ . The FLLDx bits configure the divider, D, to 1, 2, 4 or 8. By default, DCOPLUS = 0 and D = 2 providing a clock frequency of  $f_{\text{DCO}/2}$  on  $f_{\text{DCOCLK}}$ .

The multiplier (N+1) and D set the frequency of DCOCLK.

$$\begin{aligned} \text{DCOPLUS} = 0: f_{\text{DCOCLK}} &= (N + 1) \times f_{\text{ACLK}} \\ \text{DCOPLUS} = 1: f_{\text{DCOCLK}} &= D \times (N + 1) \times f_{\text{ACLK}} \end{aligned}$$

#### DCO Frequency Range

The frequency range of  $f_{\text{DCO}}$  is selected with the FNx bits as listed in Table 4–1. The range control allows the DCO to operate near the center of the available taps for a given DCOCLK frequency. The user must ensure that MCLK does not exceed the maximum operating frequency. See the device-specific datasheet for parameters.

Table 4–1. DCO Range Control Bits

FN_8	FN_4	FN_3	FN_2	Typical $f_{\text{DCO}}$ Range
0	0	0	0	0.65–6.1
0	0	0	1	1.3–12.1
0	0	1	X	2–17.9
0	1	X	X	2.8–26.6
1	X	X	X	4.2–46

#### 4.2.5 Frequency Locked Loop (FLL)

The FLL continuously counts up or down a 10-bit frequency integrator. The output of the frequency integrator that drives the DCO can be read in SCFI1 and SCFI0. The count is adjusted +1 or –1 with each ACLK crystal period.

Five of the integrator bits, SCFI1 bits 7-3, set the DCO frequency tap. Twenty-nine taps are implemented for the DCO (28, 29, 30, and 31 are equivalent), and each is approximately 10% higher than the previous. The modulator mixes two adjacent DCO frequencies to produce fractional taps. SCFI1 bits 2-0 and SCFI0 bits 1-0 are used for the modulator.

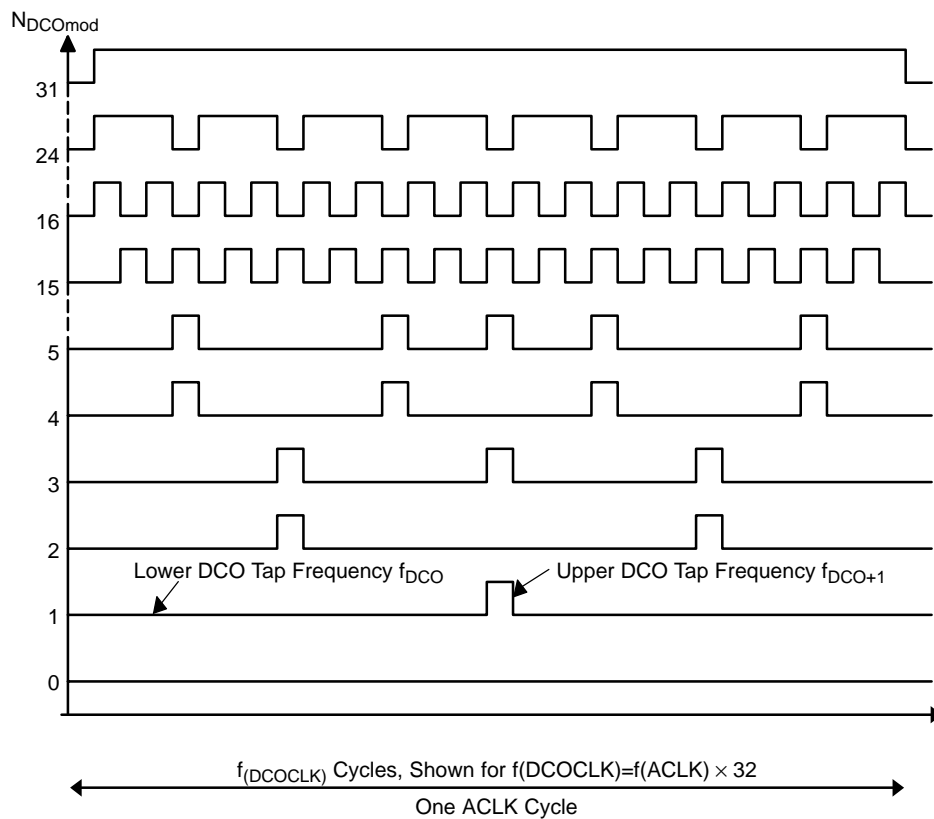
The DCO starts at the lowest tap after a PUC or when SCFI0 and SCFI1 are cleared. Time must be allowed for the DCO to settle on the proper tap for normal operation. 32 ACLK cycles are required between taps requiring a worst case of 28 x 32 ACLK cycles for the DCO to settle.

### 4.2.6 DCO Modulator

The modulator mixes two adjacent DCO frequencies to produce an intermediate effective frequency and spread the clock energy, reducing electromagnetic interference (EMI). The modulator mixes the two adjacent frequencies across 32 DCOCLK clock cycles.

The error of the effective frequency is zero every 32 DCOCLK cycles and does not accumulate. The modulator settings and DCO control are automatically controlled by the FLL hardware. Figure 4–3 illustrates the modulator operation.

Figure 4–3. Modulator Patterns



#### 4.2.7 Disabling the FLL Hardware and Modulator

The FLL is disabled when the status register bit SCG0 = 1. When the FLL is disabled, the DCO runs at the previously selected tap and DCOCLK is not automatically stabilized.

The DCO modulator is disabled when SCFQ\_M = 1. When the DCO modulator is disabled, the DCOCLK is adjusted to the nearest of the available DCO taps.

#### 4.2.8 FLL Operation from Low-Power Modes-

An interrupt service request clears SCG1, CPUOFF and OSCOFF if set but does not clear SCG0. This means that FLL operation from within an interrupt service routine entered from LPM1, 2, 3 or 4, the FLL remains disabled and the DCO operates at the previous setting as defined in SCF10 and SCF11. SCG0 can be cleared by user software if FLL operation is required.

#### 4.2.9 Buffered Clock Output

ACLK may be divided by 1, 2, 4, or 8 and buffered out of the device on P1.5. The division rate is selected with the FLL\_DIV bits.

The ACLK output is multiplexed with other pin functions. When multiplexed, the pin must be configured for the ACLK output.

```
BIS.B #P1SEL_5,&P1SEL      ; Select ACLK/n signal as
                             ; output for port P1.5 if
BIS.B #P1DIR_5,&P1DIR      ; Select port P1.5 to ACLK/n
                             ; signal for output
```

#### 4.2.10 FLL+ Fail-Safe Operation

The FLL+ module incorporates an oscillator-fault fail-safe feature. This feature detects an oscillator fault for LFXT1, DCO and XT2 as shown in Figure 4–4. The available fault conditions are:

- Low-frequency oscillator fault (LFOF) for LFXT1 in LF mode
- High-frequency oscillator fault (XT1OF) for LFXT1 in HF mode
- High-frequency oscillator fault (XT2OF) for XT2
- DCO fault flag (DCOF) for the DCO

The crystal oscillator fault bits LFOF, XT1OF and XT2OF are set if the corresponding crystal oscillator is turned on and not operating properly. The fault bits remain set as long as the fault condition exists and are automatically cleared if the enabled oscillators function normally. During a LFXT1 crystal failure, no ACLK signal is generated and the FLL+ continues to count down to zero in an attempt to lock ACLK and MCLK/(D×[N+1]). The DCO tap moves to the lowest position (SCF11.7 to SCF11.3 are cleared) and the DCOF is set. A DCOF is also generated if the N-multiplier value is set too high for the selected DCO frequency range resulting the DCO tap to move to the highest position (SCF11.7 to SCF11.3 are set). The DCOF is cleared automatically if the DCO tap is not in the lowest or the highest positions.

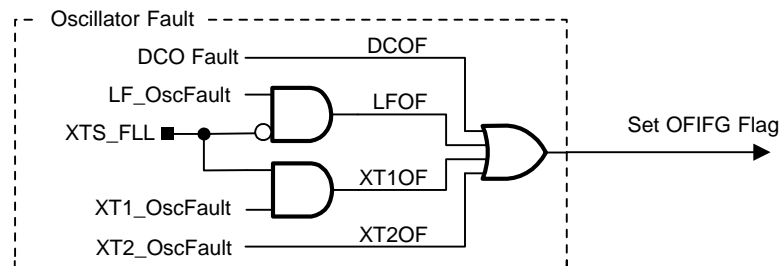
The OFIFG oscillator-fault interrupt flag is set and latched at POR or when an oscillator fault (LFOF, XT1OF, XT2OF, or DCOF set) is detected. When OFIFG is set, MCLK is sourced from the DCO, and if OFIE is set, the OFIFG requests an NMI interrupt. When the interrupt is granted, the OFIE is reset automatically. The OFIFG flag must be cleared by software. The source of the fault can be identified by checking the individual fault bits.

When OFIFG is set and MCLK is automatically switched to the DCO, the SELMx bit settings are not changed. This condition must be handled by user software.

**Note: DCO Active During Oscillator Fault**

DCOCLK is active even at the lowest DCO tap. The clock signal is available for the CPU to execute code and service an NMI during an oscillator fault.

Figure 4–4. Oscillator Fault Logic



### 4.3 FLL+ Clock Module Registers

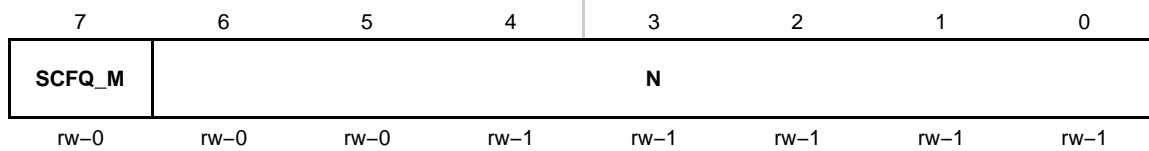
The FLL+ registers are listed in Table 4–2.

*Table 4–2. FLL+ Registers*

<b>Register</b>	<b>Short Form</b>	<b>Register Type</b>	<b>Address</b>	<b>Initial State</b>
System clock control	SCFQCTL	Read/write	052h	01Fh with PUC
System clock frequency integrator 0	SCFI0	Read/write	050h	040h with PUC
System clock frequency integrator 1	SCFI1	Read/write	051h	Reset with PUC
FLL+ control register 0	FLL_CTL0	Read/write	053h	003h with PUC
FLL+ control register 1	FLL_CTL1	Read/write	054h	Reset with PUC
SFR interrupt enable register 1	IE1	Read/write	0000h	Reset with PUC
SFR interrupt flag register 1	IFG1	Read/write	0002h	Reset with PUC

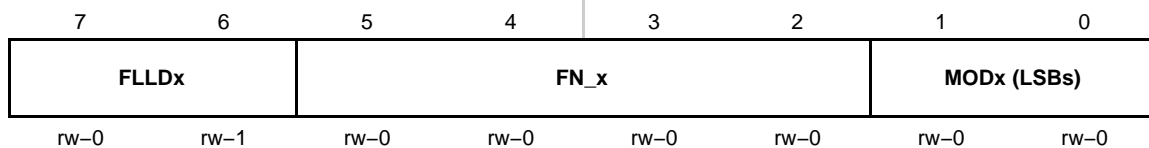


### SCFQCTL, System Clock Control Register

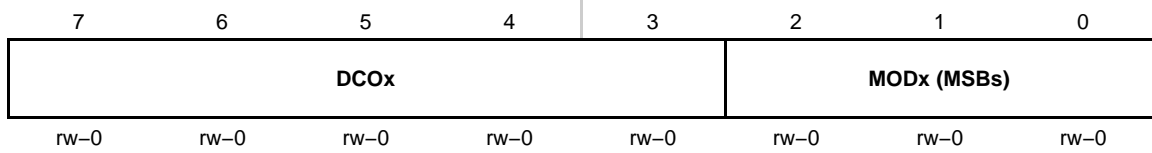


- SCFQ\_M** Bit 7 Modulation. This enables or disables modulation  
 0 Modulation enabled  
 1 Modulation disabled
- N** Bits 6-0 Multiplier. These bits set the multiplier value for the DCO. N must be > 0 or unpredictable operation will result.  
 When DCOPLUS=0:  $f_{DCOCLK} = (N + 1) \cdot f_{crystal}$   
 When DCOPLUS=1:  $f_{DCOCLK} = D \times (N + 1) \cdot f_{crystal}$

### SCFI0, System Clock Frequency Integrator Register 0



- FLLDx** Bits 7-6 FLL+ loop divider. These bits divide  $f_{DCOCLK}$  in the FLL+ feedback loop. This results in an additional multiplier for the multiplier bits. See also multiplier bits.  
 00 /1  
 01 /2  
 10 /4  
 11 /8
- FN\_x** Bits 5-2 DCO Range Control. These bits select the  $f_{DCO}$  operating range.  
 0000 0.65 - 6.1 MHz  
 0001 1.3 - 12.1 MHz  
 001x 2 - 17.9 MHz  
 01xx 2.8 - 26.6 MHz  
 1xxx 4.2 - 46 MHz
- MODx** Bits 1-0 Least significant modulator bits. Bit 0 is the modulator LSB. These bits affect the modulator pattern. All MODx bits are modified automatically by the FLL+.

**SCF11, System Clock Frequency Integrator Register 1**

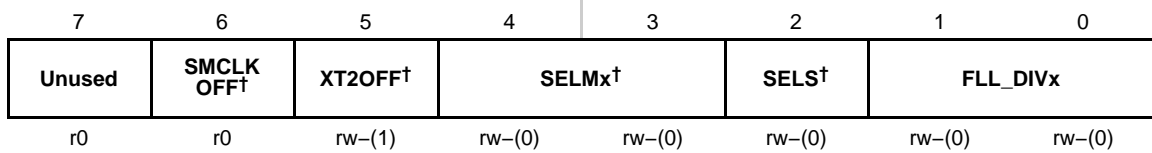
<b>DCOx</b>	Bits 7-3	These bits select the DCO tap and are modified automatically by the FLL+.
<b>MODx</b>	Bit 2	Most significant modulator bits. Bit 2 is the modulator MSB. These bits affect the modulator pattern. All MODx bits are modified automatically by the FLL+.

**FLL\_CTL0, FLL+ Control Register 0**

7	6	5	4	3	2	1	0
<b>DCOPLUS</b>	<b>XTS_FLL</b>	<b>XCAPxPF</b>		<b>XT2OF†</b>	<b>XT1OF</b>	<b>LFOF</b>	<b>DCOF</b>
rw-0	rw-0	rw-0	rw-0	r0	r0	r-(1)	r-1

† Not present in MSP430x41x, MSP430x42x devices

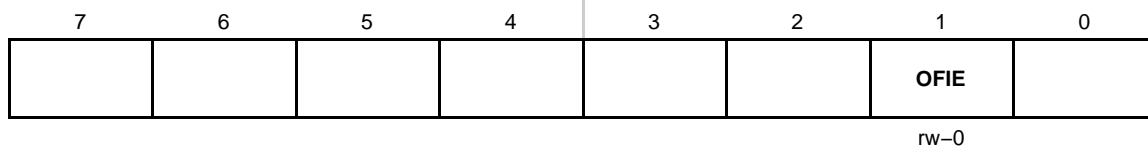
<b>DCOPLUS</b>	Bit 7	DCO output pre-divider. This bit selects if the DCO output is pre-divided before sourcing MCLK or SMCLK. The division rate is selected with the FLL_DIV bits 0 DCO output is divided 1 DCO output is not divided
<b>XTS_FLL</b>	Bit 6	LFTX1 mode select 0 Low frequency mode 1 High frequency mode
<b>XCAPxPF</b>	Bits 5-4	Oscillator capacitor selection. These bits select the effective capacitance seen by the LFXT1 crystal or resonator when XTS_FLL = 0. 00 ~1 pF 01 ~6 pF 10 ~8 pF 11 ~10 pF
<b>XT2OF</b>	Bit 3	XT2 oscillator fault. Not present in MSP430x41x, MSP430x42x devices. 0 No fault condition present 1 Fault condition present
<b>XT1OF</b>	Bit 2	LFXT1 high frequency oscillator fault 0 No fault condition present 1 Fault condition present
<b>LFOF</b>	Bit 1	LFXT1 low frequency oscillator fault 0 No fault condition present 1 Fault condition present
<b>DCOF</b>	Bit 0	DCO oscillator fault 0 No fault condition present 1 Fault condition present

**FLL\_CTL1, FLL+ Control Register 1**

† Not present in MSP430x41x, MSP430x42x devices.

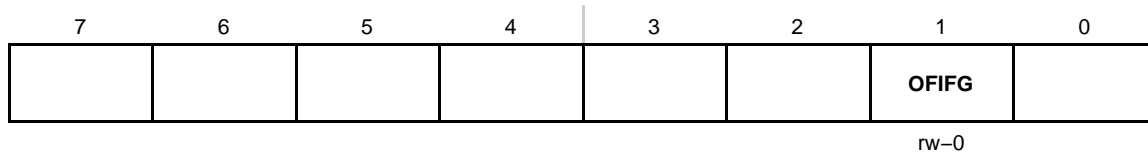
<b>Unused</b>	Bit 7	
<b>SMCLKOFF</b>	Bit 6	SMCLK off. This bit turns off SMCLK. Not present in MSP430x41x, MSPx42x devices. 0 SMCLK is on 1 SMCLK is off
<b>XT2OFF</b>	Bit 5	XT2 off. This bit turns off the XT2 oscillator. Not present in MSP430x41x, MSPx42x devices. 0 XT2 is on 1 XT2 is off if it is not used for MCLK or SMCLK.
<b>SELMx</b>	Bits 4-3	Select MCLK. These bits select the MCLK source. Not present in MSP430x41x, MSP430x42x devices. 00 DCOCLK 01 DCOCLK 10 XT2CLK 11 LFXT1CLK
<b>SELS</b>	Bit 2	Select SMCLK. This bit selects the SMCLK source. Not present in MSP430x41x, MSP430x42x devices. 0 DCOCLK 1 XT2CLK
<b>FLL_DIVx</b>	Bits 1-0	ACLK divider 00 /1 01 /2 10 /4 11 /8

### IE1, Interrupt Enable Register 1



- Bits 7-2
These bits may be used by other modules. See device-specific datasheet.
- OFIE**
Bit 1
Oscillator fault interrupt enable. This bit enables the OFIFG interrupt. Because other bits in IE1 may be used for other modules, it is recommended to set or clear this bit using `BIS.B` or `BIC.B` instructions, rather than `MOV.B` or `CLR.B` instructions.
  - 0 Interrupt not enabled
  - 1 Interrupt enabled
- Bits 0
This bit may be used by other modules. See device-specific datasheet.

### IFG1, Interrupt Flag Register 1



- Bits 7-2
These bits may be used by other modules. See device-specific datasheet.
- OFIFG**
Bit 1
Oscillator fault interrupt flag. Because other bits in IFG1 may be used for other modules, it is recommended to set or clear this bit using `BIS.B` or `BIC.B` instructions, rather than `MOV.B` or `CLR.B` instructions.
  - 0 No interrupt pending
  - 1 Interrupt pending
- Bits 0
This bit may be used by other modules. See device-specific datasheet.

# Flash Memory Controller

---

---

---

---

This chapter describes the operation of the MSP430 flash memory controller.

<b>Topic</b>	<b>Page</b>
<b>5.1 Flash Memory Introduction</b> .....	<b>5-2</b>
<b>5.2 Flash Memory Segmentation</b> .....	<b>5-3</b>
<b>5.3 Flash Memory Operation</b> .....	<b>5-4</b>
<b>5.4 Flash Memory Registers</b> .....	<b>5-17</b>

## 5.1 Flash Memory Introduction

The MSP430 flash memory is bit-, byte-, and word-addressable and programmable. The flash memory module has an integrated controller that controls programming and erase operations. The controller has three registers, a timing generator, and a voltage generator to supply program and erase voltages.

MSP430 flash memory features include:

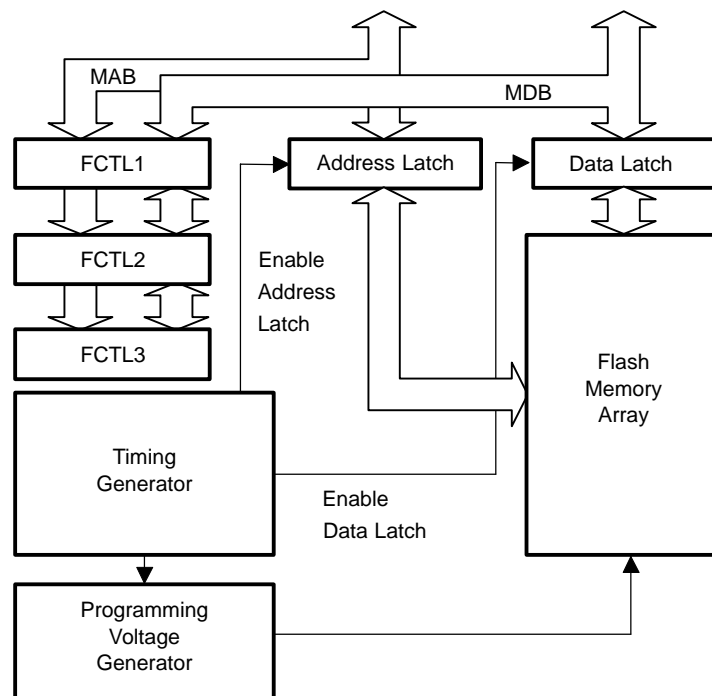
- Internal programming voltage generation
- Bit, byte or word programmable
- Ultralow-power operation
- Segment erase and mass erase

The block diagram of the flash memory and controller is shown in Figure 5–1.

**Note: Minimum  $V_{CC}$  During Flash Write or Erase**

The minimum  $V_{CC}$  voltage during a flash write or erase operation is 2.7 V. If  $V_{CC}$  falls below 2.7 V during a write or erase, the result of the write or erase will be unpredictable.

Figure 5–1. Flash Memory Module Block Diagram



## 5.2 Flash Memory Segmentation

MSP430 flash memory is partitioned into segments. Single bits, bytes, or words can be written to flash memory, but the segment is the smallest size of flash memory that can be erased.

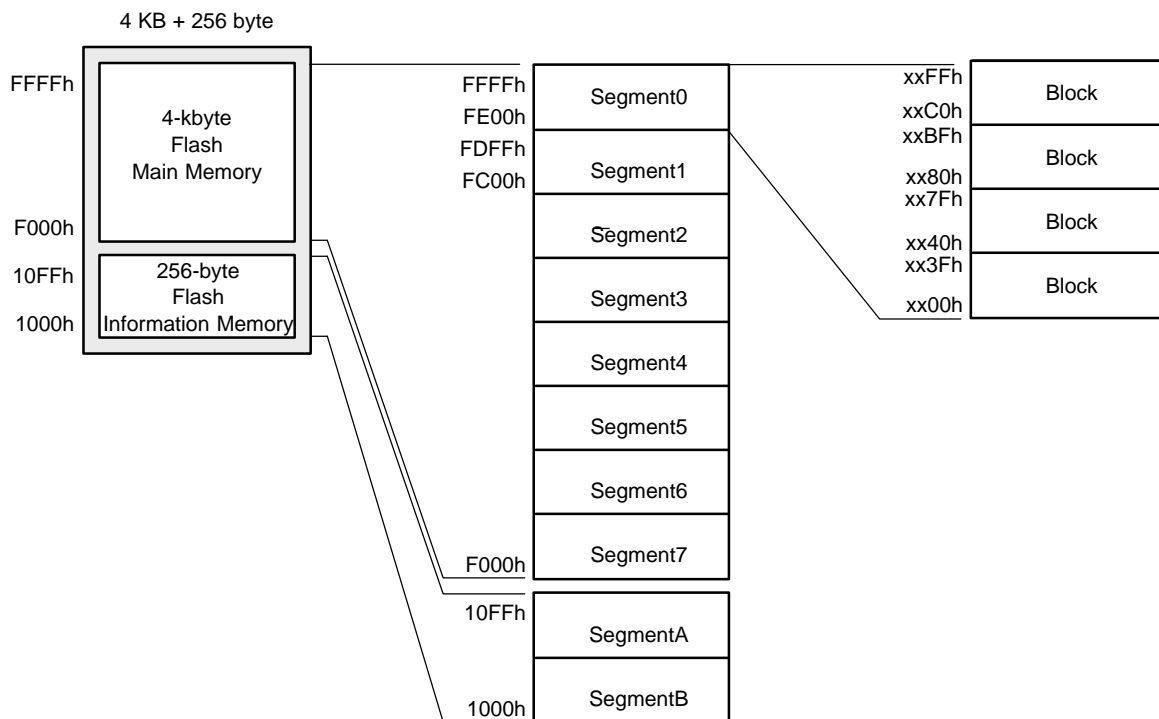
The flash memory is partitioned into main and information memory sections. There is no difference in the operation of the main and information memory sections. Code or data can be located in either section. The differences between the two sections are the segment size and the physical addresses.

The information memory has two 128-byte segments. The main memory has two or more 512-byte segments. See the device-specific datasheet for the complete memory map of a device.

The segments are further dividing into blocks. A block is 64 bytes, starting at 0xx00h, 0xx40h, 0xx80h, or 0xxC0h, and ending at 0xx3Fh, 0xx7Fh, 0xxBFh, or 0xFFh.

Figure 5–2 shows the flash segmentation using an example of 4-KB flash that has eight main segments and both information segments.

Figure 5–2. Flash Memory Segments, 4-KB Example





### 5.3 Flash Memory Operation

The default mode of the flash memory is read mode. In read mode, the flash memory is not being erased or written, the flash timing generator and voltage generator are off, and the memory operates identically to ROM.

MSP430 flash memory is in-system programmable (ISP) without the need for additional external voltage. The CPU can program its own flash memory. The flash memory write/erase modes are selected with the BLKWRT, WRT, MERAS, and ERASE bits and are:

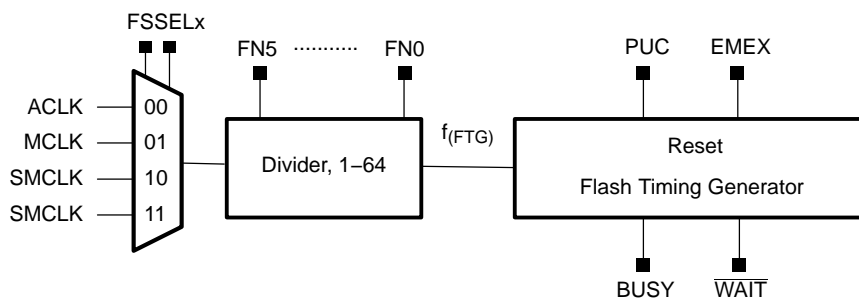
- Byte/word write
- Block write
- Segment Erase
- Mass Erase (all main memory segments)
- All Erase (all segments)

Reading or writing to flash memory while it is being programmed or erased is prohibited. If CPU execution is required during the write or erase, the code to be executed must be in RAM. Any flash update can be initiated from within flash memory or RAM.

#### 5.3.1 Flash Memory Timing Generator

Write and erase operations are controlled by the flash timing generator shown in Figure 5–3. The flash timing generator operating frequency,  $f_{FTG}$ , must be in the range from ~ 257 kHz to ~ 476 kHz (see device-specific datasheet).

Figure 5–3. Flash Memory Timing Generator Block Diagram



The flash timing generator can be sourced from ACLK, SMCLK, or MCLK. The selected clock source should be divided using the FNx bits to meet the frequency requirements for  $f_{FTG}$ . If the  $f_{FTG}$  frequency deviates from the specification during the write or erase operation, the result of the write or erase may be unpredictable, or the flash memory may be stressed above the limits of reliable operation.

### 5.3.2 Erasing Flash Memory

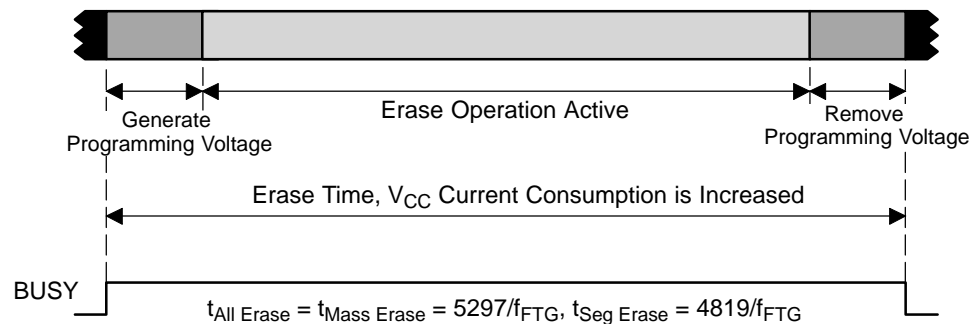
The erased level of a flash memory bit is 1. Each bit can be programmed from 1 to 0 individually but to reprogram from 0 to 1 requires an erase cycle. The smallest amount of flash that can be erased is a segment. There are three erase modes selected with the ERASE and MERAS bits listed in Table 5–1.

Table 5–1. Erase Modes

MERAS	ERASE	Erase Mode
0	1	Segment erase
1	0	Mass erase (all main memory segments)
1	1	Erase all flash memory (main and information segments)

Any erase is initiated by a dummy write into the address range to be erased. The dummy write starts the flash timing generator and the erase operation. Figure 5–4 shows the erase cycle timing. The BUSY bit is set immediately after the dummy write and remains set throughout the erase cycle. BUSY, MERAS, and ERASE are automatically cleared when the cycle completes. The erase cycle timing is not dependent on the amount of flash memory present on a device. Erase cycle times are equivalent for all MSP430F4xx devices.

Figure 5–4. Erase Cycle Timing



A dummy write to an address not in the range to be erased does not start the erase cycle, does not affect the flash memory, and is not flagged in any way. This errant dummy write is ignored.

Interrupts are automatically disabled before a flash erase cycle. After the erase cycle has completed, interrupts are automatically re-enabled. Any interrupt that occurred during the erase cycle will have its associated flag set, and will generate an interrupt request when re-enabled.

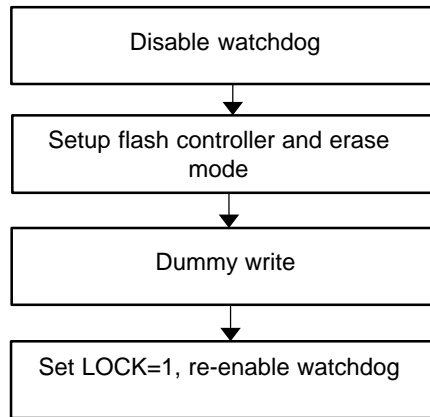
## Initiating an Erase from Within Flash Memory

Any erase cycle can be initiated from within flash memory or from RAM. When a flash segment erase operation is initiated from within flash memory, all timing is controlled by the flash controller, and the CPU is held while the erase cycle completes. After the erase cycle completes, the CPU resumes code execution with the instruction following the dummy write.

When initiating an erase cycle from within flash memory, it is possible to erase the code needed for execution after the erase. If this occurs, CPU execution will be unpredictable after the erase cycle.

The flow to initiate an erase from flash is shown in Figure 5–5.

Figure 5–5. Erase Cycle from Within Flash Memory



```

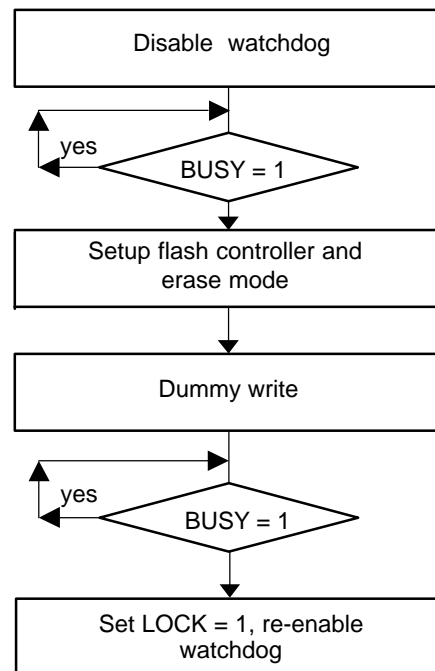
; Segment Erase from flash. 514 kHz < SMCLK < 952 kHz
; Assumes ACCVIE = NMIIE = OFIE = 0.
MOV    #WDTPW+WDTHOLD,&WDTCTL    ; Disable WDT
MOV    #FWKEY+FSSEL1+FN0,&FCTL2  ; SMCLK/2
MOV    #FWKEY,&FCTL3             ; Clear LOCK
MOV    #FWKEY+ERASE,&FCTL1       ; Enable segment erase
CLR    &0FC10h                   ; Dummy write, erase S1
MOV    #FWKEY+LOCK,&FCTL3        ; Done, set LOCK
...
; Re-enable WDT?
  
```

## Initiating an Erase from RAM

Any erase cycle may be initiated from RAM. In this case, the CPU is not held and can continue to execute code from RAM. The BUSY bit must be polled to determine the end of the erase cycle before the CPU can access any flash address again. If a flash access occurs while BUSY=1, it is an access violation, ACCVIFG will be set, and the erase results will be unpredictable.

The flow to initiate an erase from flash from RAM is shown in Figure 5–6.

Figure 5–6. Erase Cycle from Within RAM



```

; Segment Erase from RAM. 514 kHz < SMCLK < 952 kHz
; Assumes ACCVIE = NMIIE = OFIE = 0.
MOV    #WDTPW+WDTHOLD,&WDTCTL    ; Disable WDT
L1 BIT  #BUSY,&FCTL3              ; Test BUSY
JNZ    L1                        ; Loop while busy
MOV    #FWKEY+FSSEL1+FN0,&FCTL2  ; SMCLK/2
MOV    #FWKEY,&FCTL3             ; Clear LOCK
MOV    #FWKEY+ERASE,&FCTL1       ; Enable erase
CLR    &0FC10h                  ; Dummy write, erase S1
L2 BIT  #BUSY,&FCTL3              ; Test BUSY
JNZ    L2                        ; Loop while busy
MOV    #FWKEY+LOCK,&FCTL3        ; Done, set LOCK
...
; Re-enable WDT?
  
```

### 5.3.3 Writing Flash Memory

The write modes, selected by the WRT and BLKWRT bits, are listed in Table 5–1. Interrupts are automatically disabled during a flash write and re-enabled after the write. Any interrupt that occurred during the write will have its associated flag set, and will generate an interrupt request when re-enabled.

Table 5–2. Write Modes

BLKWRT	WRT	Write Mode
0	1	Byte/word write
1	1	Block write

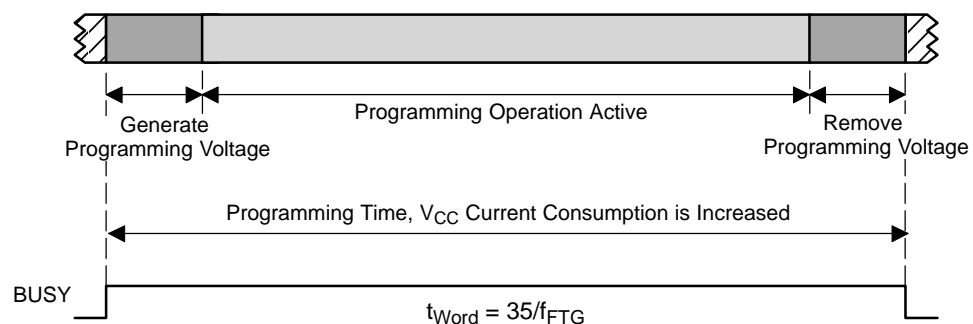
Both write modes use a sequence of individual write instructions, but using the block write mode is approximately twice as fast as byte/word mode, because the voltage generator remains on for the complete block write. Any instruction that modifies a destination can be used to modify a flash location in either byte/word write mode or block write mode.

The BUSY bit is set while a write operation is active and cleared when the operation completes. If the write operation is initiated from RAM, the CPU must not access flash while BUSY=1. Otherwise, an access violation occurs, ACCVIFG is set, and the flash write is unpredictable.

#### Byte/Word Write

A byte/word write operation can be initiated from within flash memory or from RAM. When initiating from within flash memory, all timing is controlled by the flash controller, and the CPU is held while the write completes. After the write completes, the CPU resumes code execution with the instruction following the write. The byte/word write timing is shown in Figure 5–7.

Figure 5–7. Byte/Word Write Timing



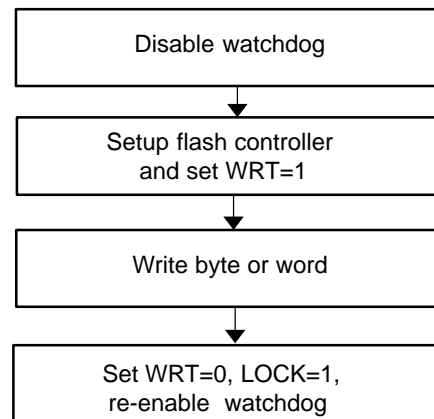
When a byte/word write is executed from RAM, the CPU continues to execute code from RAM. The BUSY bit must be zero before the CPU accesses flash again, otherwise an access violation occurs, ACCVIFG is set, and the write result is unpredictable.

In byte/word mode, the internally-generated programming voltage is applied to the complete 64-byte block, each time a byte or word is written, for 32 of the 35  $f_{FTG}$  cycles. With each byte or word write, the amount of time the block is subjected to the programming voltage accumulates. The cumulative programming time,  $t_{CPT}$ , must not be exceeded for any block. If the cumulative programming time is met, the block must be erased before performing any further writes to any address within the block. See the device-specific datasheet for specifications.

### Initiating a Byte/Word Write from Within Flash Memory

The flow to initiate a byte/word write from flash is shown in Figure 5–8.

Figure 5–8. Initiating a Byte/Word Write from Flash



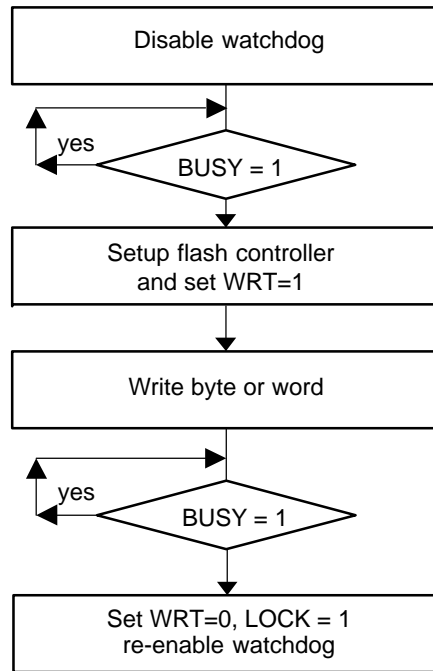
```

; Byte/word write from flash. 514 kHz < SMCLK < 952 kHz
; Assumes 0FF1Eh is already erased
; Assumes ACCVIE = NMIE = OFIE = 0.
MOV    #WDPW+WDTHOLD,&WDTCTL    ; Disable WDT
MOV    #FWKEY+FSSEL1+FN0,&FCTL2  ; SMCLK/2
MOV    #FWKEY,&FCTL3             ; Clear LOCK
MOV    #FWKEY+WRT,&FCTL1         ; Enable write
MOV    #0123h,&0FF1Eh           ; 0123h  -> 0FF1Eh
MOV    #FWKEY,&FCTL1             ; Done. Clear WRT
MOV    #FWKEY+LOCK,&FCTL3       ; Set LOCK
...
; Re-enable WDT?
  
```

## Initiating a Byte/Word Write from RAM

The flow to initiate a byte/word write from RAM is shown in Figure 5–9.

Figure 5–9. Initiating a Byte/Word Write from RAM



```

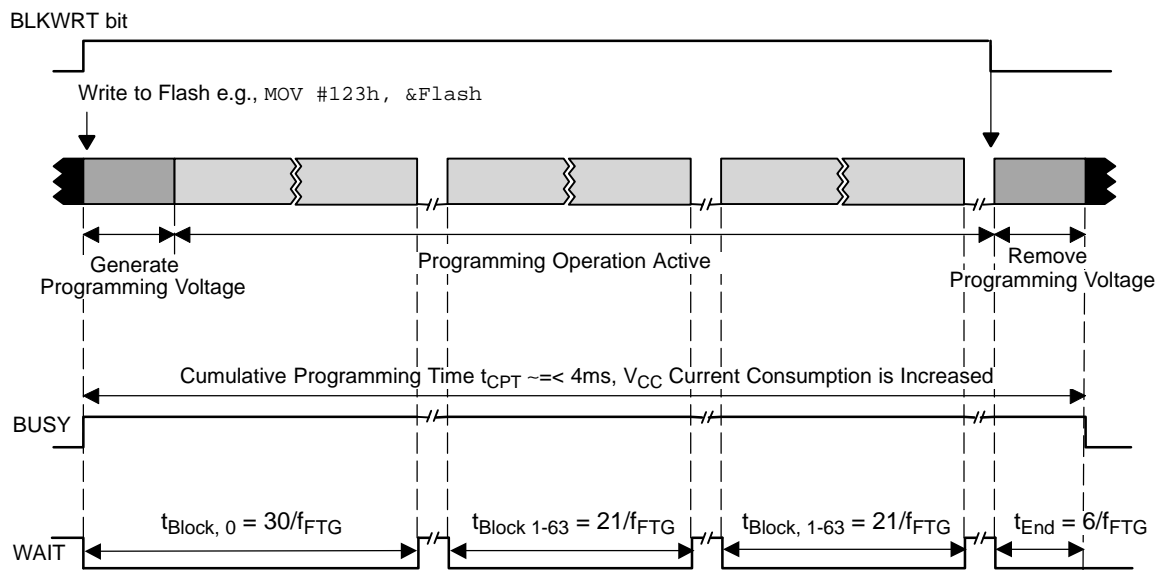
; Byte/word write from RAM. 514 kHz < SMCLK < 952 kHz
; Assumes 0FF1Eh is already erased
; Assumes ACCVIE = NMIIE = OFIE = 0.
MOV #WDTPW+WDTHOLD,&WDTCTL ; Disable WDT
L1 BIT #BUSY,&FCTL3 ; Test BUSY
JNZ L1 ; Loop while busy
MOV #FWKEY+FSSEL1+FN0,&FCTL2 ; SMCLK/2
MOV #FWKEY,&FCTL3 ; Clear LOCK
MOV #FWKEY+WRT,&FCTL1 ; Enable write
MOV #0123h,&0FF1Eh ; 0123h -> 0FF1Eh
L2 BIT #BUSY,&FCTL3 ; Test BUSY
JNZ L2 ; Loop while busy
MOV #FWKEY,&FCTL1 ; Clear WRT
MOV #FWKEY+LOCK,&FCTL3 ; Set LOCK
... ; Re-enable WDT?
  
```

## Block Write

The block write can be used to accelerate the flash write process when many sequential bytes or words need to be programmed. The flash programming voltage remains on for the duration of writing the 64-byte block. The cumulative programming time  $t_{CPT}$  must not be exceeded for any block during a block write.

A block write cannot be initiated from within flash memory. The block write must be initiated from RAM only. The BUSY bit remains set throughout the duration of the block write. The WAIT bit must be checked between writing each byte or word in the block. When WAIT is set the next byte or word of the block can be written. When writing successive blocks, the BLKWRT bit must be cleared after the current block is complete. BLKWRT can be set initiating the next block write after the required flash recovery time given by  $t_{End}$ . BUSY is cleared following each block write completion indicating the next block can be written. Figure 5–10 shows the block write timing.

Figure 5–10. Block-Write Cycle Timing

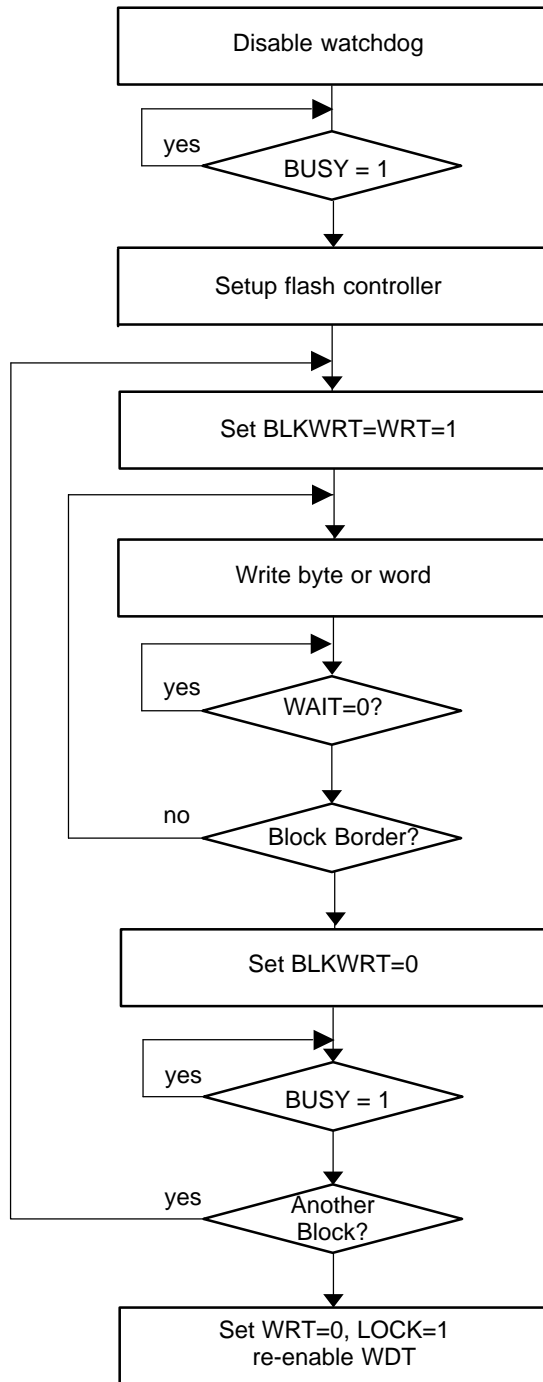




### Block Write Flow and Example

A block write flow is shown in Figure 5–8 and the following example.

Figure 5–11. Block Write Flow



```

; Write one block starting at 0F000h.
; Must be executed from RAM, Assumes Flash is already erased.
; 514 kHz < SMCLK < 952 kHz
; Assumes ACCVIE = NMIIE = OFIE = 0.
    MOV    #32,R5                ; Use as write counter
    MOV    #0F000h,R6           ; Write pointer
    MOV    #WDTPW+WDTHOLD,&WDTCTL ; Disable WDT
L1 BIT    #BUSY,&FCTL3           ; Test BUSY
    JNZ    L1                    ; Loop while busy
    MOV    #FWKEY+FSSEL1+FN0,&FCTL2 ; SMCLK/2
    MOV    #FWKEY,&FCTL3         ; Clear LOCK
    MOV    #FWKEY+BLKWRT+WRT,&FCTL1 ; Enable block write
L2 MOV    Write_Value,0(R6)     ; Write location
L3 BIT    #WAIT,&FCTL3          ; Test WAIT
    JZ     L3                    ; Loop while WAIT=0
    INCD   R6                    ; Point to next word
    DEC    R5                    ; Decrement write counter
    JNZ    L2                    ; End of block?
    MOV    #FWKEY,&FCTL1         ; Clear WRT,BLKWRT
L4 BIT    #BUSY,&FCTL3           ; Test BUSY
    JNZ    L4                    ; Loop while busy
    MOV    #FWKEY+LOCK,&FCTL3    ; Set LOCK
    ...                          ; Re-enable WDT if needed

```

### 5.3.4 Flash Memory Access During Write or Erase

When any write or any erase operation is initiated from RAM and while  $BUSY=1$ , the CPU may not read or write to or from any flash location. Otherwise, an access violation occurs,  $ACCVIFG$  is set, and the result is unpredictable. Also if a write to flash is attempted with  $WRT=0$ , the  $ACCVIFG$  interrupt flag is set, and the flash memory is unaffected.

When a byte/word write or any erase operation is initiated from within flash memory, the flash controller returns op-code  $03FFFh$  to the CPU at the next instruction fetch. Op-code  $03FFFh$  is the  $JMP\ PC$  instruction. This causes the CPU to loop until the flash operation is finished. When the operation is finished and  $BUSY=0$ , the flash controller allows the CPU to fetch the proper op-code and program execution resumes.

The flash access conditions while  $BUSY=1$  are listed in Table 5–3.

Table 5–3. Flash Access While  $BUSY = 1$

Flash Operation	Flash Access	WAIT	Result
Any erase, or Byte/word write	Read	0	$ACCVIFG = 0$ . $03FFFh$ is the value read
	Write	0	$ACCVIFG = 1$ . Write is ignored
	Instruction fetch	0	$ACCVIFG = 0$ . CPU fetches $03FFFh$ . This is the $JMP\ PC$ instruction.
Block write	Any	0	$ACCVIFG = 1$ , $LOCK = 1$
	Read	1	$ACCVIFG = 0$ , $03FFFh$ is the value read
	Write	1	$ACCVIFG = 0$ , Write is ignored
	Instruction fetch	1	$ACCVIFG = 1$ , $LOCK = 1$

### 5.3.5 Stopping a Write or Erase Cycle

Any write or erase operation can be stopped before its normal completion by setting the emergency exit bit EMEX. Setting the EMEX bit stops the active operation immediately and stops the flash controller. All flash operations cease, the flash returns to read mode, and all bits in the FCTL1 register are reset. The result of the intended operation is unpredictable.

### 5.3.6 Configuring and Accessing the Flash Memory Controller

The FCTLx registers are 16-bit, password-protected, read/write registers. Any read or write access must use word instructions and write accesses must include the write password 0A5h in the upper byte. Any write to any FCTLx register with any value other than 0A5h in the upper byte is a security key violation, sets the KEYV flag and triggers a PUC system reset. Any read of any FCTLx registers reads 096h in the upper byte.

Any write to FCTL1 during an erase or byte/word write operation is an access violation and sets ACCVIFG. Writing to FCTL1 is allowed in block write mode when WAIT=1, but writing to FCTL1 in block write mode when WAIT=0 is an access violation and sets ACCVIFG.

Any write to FCTL2 when the BUSY=1 is an access violation.

Any FCTLx register may be read when BUSY=1. A read will not cause an access violation.

### 5.3.7 Flash Memory Controller Interrupts

The flash controller has two interrupt sources, KEYV, and ACCVIFG. ACCVIFG is set when an access violation occurs. When the ACCVIE bit is re-enabled after a flash write or erase, a set ACCVIFG flag will generate an interrupt request. ACCVIFG sources the NMI interrupt vector, so it is not necessary for GIE to be set for ACCVIFG to request an interrupt. ACCVIFG may also be checked by software to determine if an access violation occurred. ACCVIFG must be reset by software.

The key violation flag KEYV is set when any of the flash control registers are written with an incorrect password. When this occurs, a PUC is generated immediately resetting the device.

### 5.3.8 Programming Flash Memory Devices

There are three options for programming an MSP430 flash device. All options support in-system programming:

- Program via JTAG
- Program via the Bootstrap Loader
- Program via a custom solution

## Programming Flash Memory via JTAG

MSP430 devices can be programmed via the JTAG port. The JTAG interface requires four signals (5 signals on 20- and 28-pin devices), ground and optionally  $V_{CC}$  and  $\overline{RST/NMI}$ .

The JTAG port is protected with a fuse. Blowing the fuse completely disables the JTAG port and is not reversible. Further access to the device via JTAG is not possible. For more details see the Application report *Programming a Flash-Based MSP430 Using the JTAG Interface* at [www.msp430.com](http://www.msp430.com).

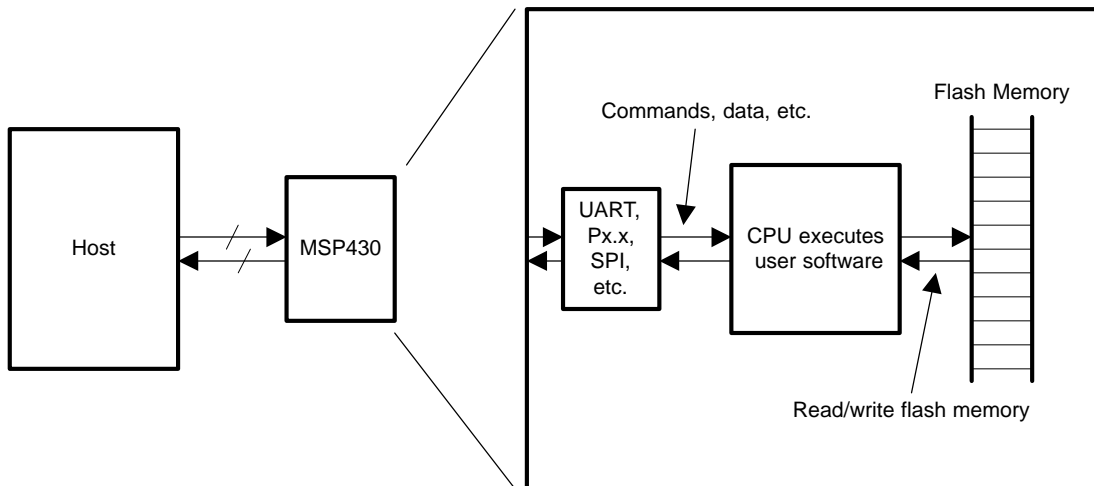
## Programming Flash Memory via the Bootstrap loader (BSL)

Every MSP430 flash device contains a bootstrap loader. The BSL enables users to read or program the flash memory or RAM using a UART serial interface. Access to the MSP430 flash memory via the BSL is protected by a 256-bit, user-defined password. For more details see the Application report *Features of the MSP430 Bootstrap Loader* at [www.msp430.com](http://www.msp430.com).

## Programming Flash Memory via a Custom Solution

The ability of the MSP430 CPU to write to its own flash memory allows for in-system and external custom programming solutions as shown in Figure 5–12. The user can choose to provide data to the MSP430 through any means available (UART, SPI, etc.). User-developed software can receive the data and program the flash memory. Since this type of solution is developed by the user, it can be completely customized to fit the application needs for programming, erasing, or updating the flash memory.

Figure 5–12. User-Developed Programming Solution



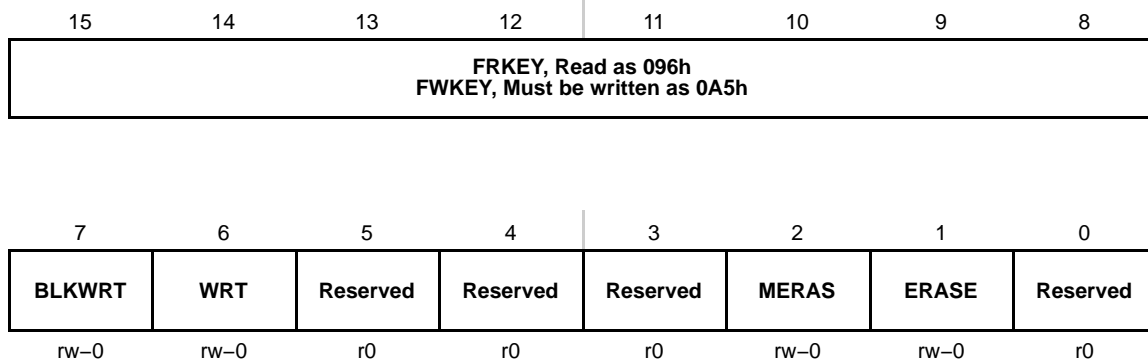
## 5.4 Flash Memory Registers

The flash memory registers are listed in Table 5–4.

*Table 5–4. Flash Memory Registers*

<b>Register</b>	<b>Short Form</b>	<b>Register Type</b>	<b>Address</b>	<b>Initial State</b>
Flash memory control register 1	FCTL1	Read/write	0128h	09600h with PUC
Flash memory control register 2	FCTL2	Read/write	012Ah	09642h with PUC
Flash memory control register 3	FCTL3	Read/write	012Ch	09618h with PUC
Interrupt Enable 1	IE1	Read/write	000h	Reset with PUC

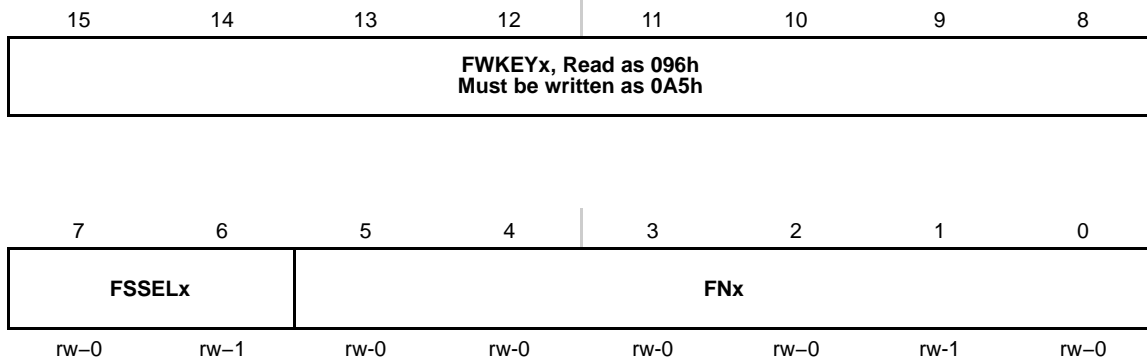
## FCTL1, Flash Memory Control Register



<b>FRKEY/ FWKEY</b>	Bits 15-8	FCTLx password. Always read as 096h. Must be written as 0A5h or a PUC will be generated.
<b>BLKWRT</b>	Bit 7	Block write mode. WRT must also be set for block write mode. BLKWRT is automatically reset when EMEX is set. 0 Block-write mode is off 1 Block-write mode is on
<b>WRT</b>	Bit 6	Write. This bit is used to select any write mode. WRT is automatically reset when EMEX is set. 0 Write mode is off 1 Write mode is on
<b>Reserved</b>	Bits 5-3	Reserved. Always read as 0.
<b>MERAS ERASE</b>	Bit 2 Bit 1	Mass erase and erase. These bits are used together to select the erase mode. MERAS and ERASE are automatically reset when EMEX is set.

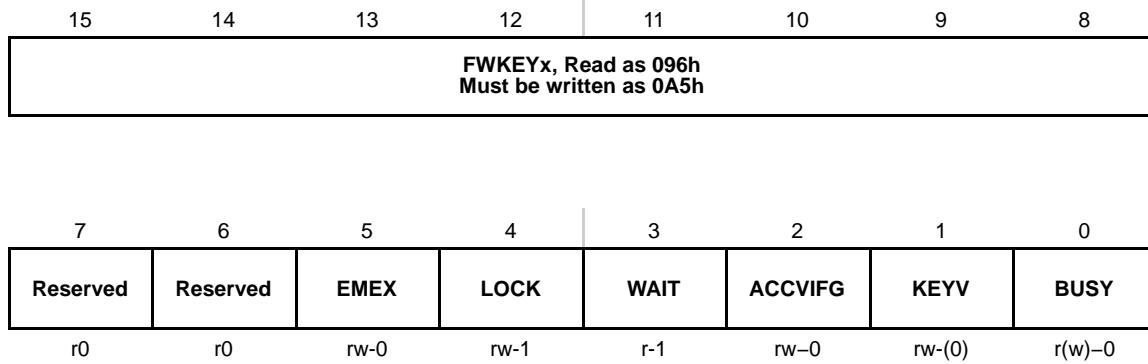
MERAS	ERASE	Erase Cycle
0	0	No erase
0	1	Erase individual segment only
1	0	Erase all main memory segments
1	1	Erase all main and information memory segments

<b>Reserved</b>	Bit 0	Reserved. Always read as 0.
-----------------	-------	-----------------------------

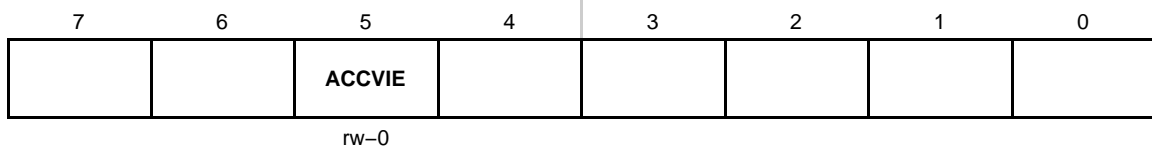
**FCTL2, Flash Memory Control Register**

<b>FWKEYx</b>	Bits 15-8	FCTLx password. Always read as 096h. Must be written as 0A5h or a PUC will be generated.
<b>FSELx</b>	Bits 7-6	Flash controller clock source select 00 ACLK 01 MCLK 10 SMCLK 11 SMCLK
<b>FNx</b>	Bits 5-0	Flash controller clock divider. These six bits select the divider for the flash controller clock. The divisor value is FNx + 1. For example, when FNx=00h, the divisor is 1. When FNx=03Fh the divisor is 64.



**FCTL3, Flash Memory Control Register FCTL3**

<b>FWKEYx</b>	Bits 15-8	FCTLx password. Always read as 096h. Must be written as 0A5h or a PUC will be generated.
<b>Reserved</b>	Bits 7-6	Reserved. Always read as 0.
<b>EMEX</b>	Bit 5	Emergency exit 0 No emergency exit 1 Emergency exit
<b>LOCK</b>	Bit 4	Lock. This bit unlocks the flash memory for writing or erasing. The LOCK bit can be set anytime during a byte/word write or erase operation and the operation will complete normally. In the block write mode if the LOCK bit is set while BLKWRT=WAIT=1, then BLKWRT and WAIT are reset and the mode ends normally. 0 Unlocked 1 Locked
<b>WAIT</b>	Bit 3	Wait. Indicates the flash memory is being written to. 0 The flash memory is not ready for the next byte/word write 1 The flash memory is ready for the next byte/word write
<b>ACCVIFG</b>	Bit 2	Access violation interrupt flag 0 No interrupt pending 1 Interrupt pending
<b>KEYV</b>	Bit 1	Flash security key violation. This bit indicates an incorrect FCTLx password was written to any flash control register and generates a PUC when set. KEYV must be reset with software. 0 FCTLx password was written correctly 1 FCTLx password was written incorrectly
<b>BUSY</b>	Bit 0	Busy. This bit indicates the status of the flash timing generator. 0 Not Busy 1 Busy

**IE1, Interrupt Enable Register 1**

Bits 7-6, 4-0 These bits may be used by other modules. See device-specific datasheet.

**ACCVIE** Bit 5 Flash memory access violation interrupt enable. This bit enables the ACCVIFG interrupt. Because other bits in IE1 may be used for other modules, it is recommended to set or clear this bit using `BIS .B` or `BIC .B` instructions, rather than `MOV .B` or `CLR .B` instructions.

0 Interrupt not enabled  
1 Interrupt enabled

# Supply Voltage Supervisor

---

---

---

---

This chapter describes the operation of the SVS. The SVS is implemented in all MSP430x4x devices.

<b>Topic</b>	<b>Page</b>
<b>6.1 SVS Introduction</b> .....	<b>6-2</b>
<b>6.2 SVS Operation</b> .....	<b>6-4</b>
<b>6.3 SVS Registers</b> .....	<b>6-7</b>

## 6.1 SVS Introduction

The supply voltage supervisor (SVS) is used to monitor the  $AV_{CC}$  supply voltage or an external voltage. The SVS can be configured to set a flag or generate a POR reset when the supply voltage or external voltage drops below a user-selected threshold.

The SVS features include:

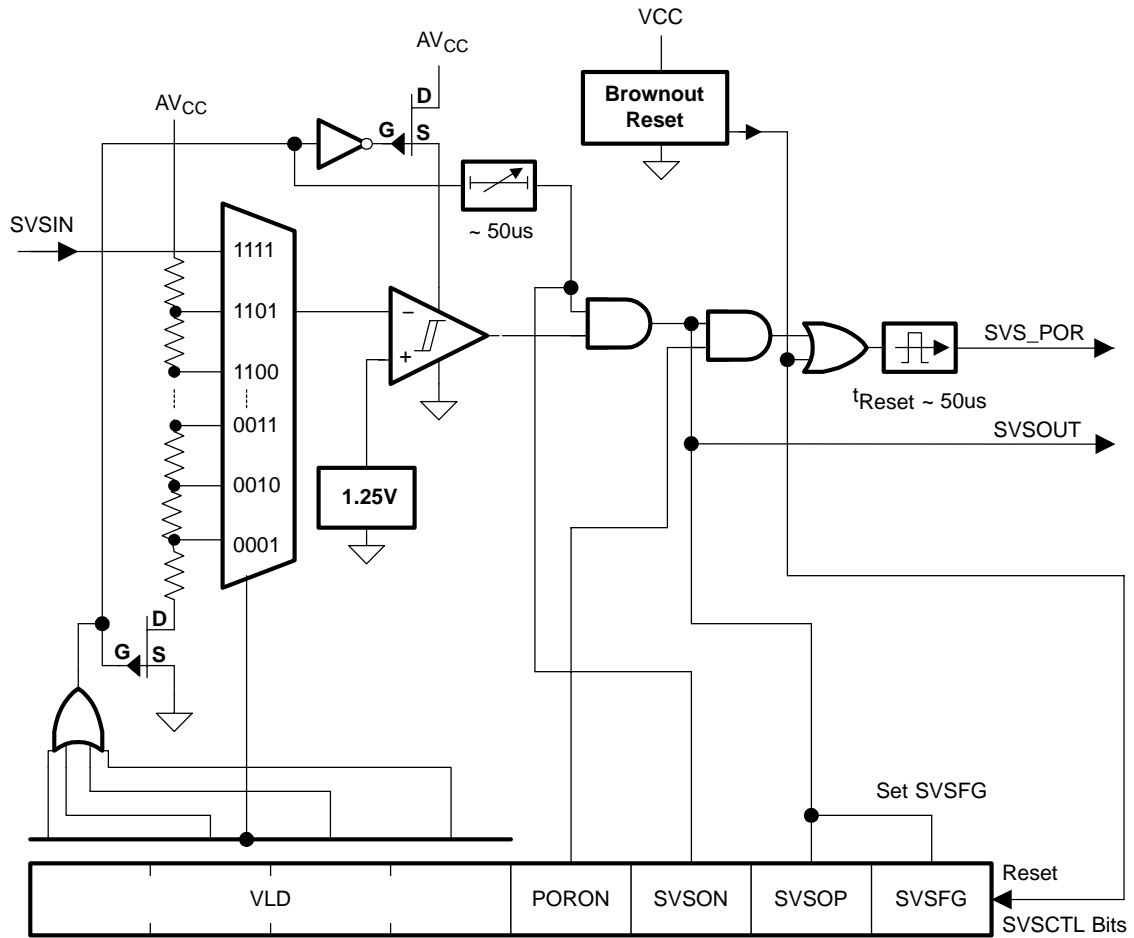
- $AV_{CC}$  monitoring
- Selectable generation of POR
- Output of SVS comparator accessible by software
- Low-voltage condition latched and accessible by software
- 14 selectable threshold levels
- External channel to monitor external voltage

The SVS block diagram is shown in Figure 6–1.

**Note: MSP430x412 and MSP430x413 Voltage Level Detect**

The MSP430x412 and MSP430x413 devices implement only one voltage level detect setting. When  $VLDx = 0$  the SVS is off. Any value greater than 0 for  $VLDx$  selects a voltage level detect of 1.9V.

Figure 6-1. SVS Block Diagram



## 6.2 SVS Operation

The SVS detects if the  $AV_{CC}$  voltage drops below a selectable level. It can be configured to provide a POR or set a flag, when a low-voltage condition occurs. The SVS is disabled after a brownout reset to conserve current consumption.

### 6.2.1 Configuring the SVS

The  $VLDx$  bits are used to enable/disable the SVS and select one of 14 threshold levels ( $V_{(SVS\_IT-)}$ ) for comparison with  $AV_{CC}$ . The SVS is off when  $VLDx = 0$  and on when  $VLDx > 0$ . The  $SVSON$  bit does not turn on the SVS. Instead, it reflects the on/off state of the SVS and can be used to determine when the SVS is on.

When  $VLDx = 1111$ , the external  $SVSIN$  channel is selected. The voltage on  $SVSIN$  is compared to an internal level of approximately 1.2 V.

### 6.2.2 SVS Comparator Operation

A low-voltage condition exists when  $AV_{CC}$  drops below the selected threshold or when the external voltage drops below its 1.2-V threshold. Any low-voltage condition sets the  $SVSFG$  bit.

The  $PORON$  bit enables or disables the device-reset function of the SVS. If  $PORON = 1$ , a POR is generated when  $SVSFG$  is set. If  $PORON = 0$ , a low-voltage condition sets  $SVSFG$ , but does not generate a POR.

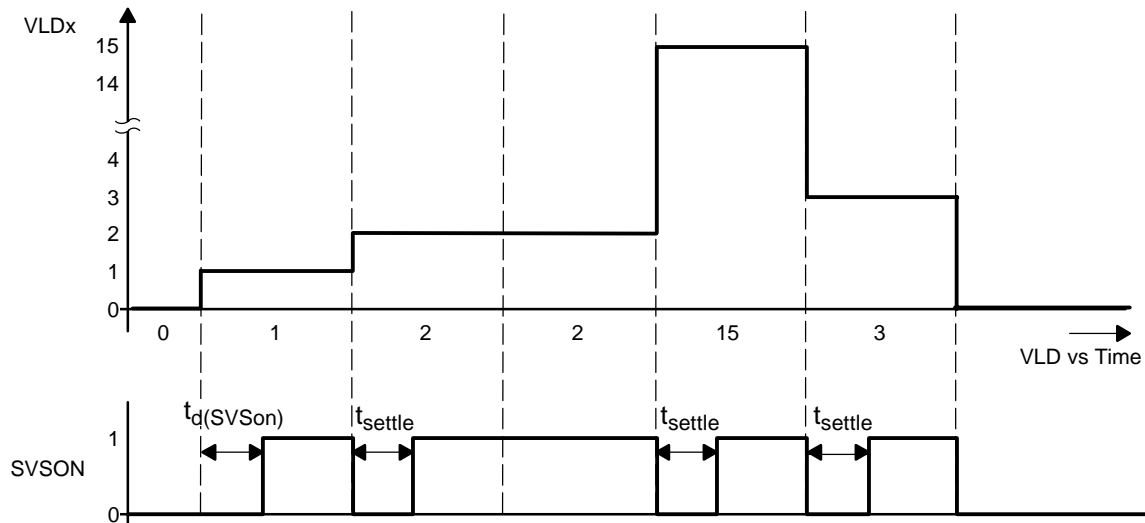
The  $SVSFG$  bit is latched. This allows user software to determine if a low-voltage condition occurred previously. The  $SVSFG$  bit must be reset by user software. If the low-voltage condition is still present when  $SVSFG$  is reset, it will be immediately set again by the SVS.

### 6.2.3 Changing the VLDx Bits

When the VLDx bits are changed, two settling delays are implemented to allow the SVS circuitry to settle. During each delay, the SVS will not set SVSFG. The delays,  $t_{d(SVSON)}$  and  $t_{settle}$ , are shown in Figure 6–2. The  $t_{d(SVSON)}$  delay takes effect when VLDx is changed from zero to any non-zero value and is approximately 50  $\mu\text{s}$ . The  $t_{settle}$  delay takes effect when the VLDx bits change from any non-zero value to any other non-zero value and is a maximum of  $\sim 12 \mu\text{s}$ . See the device-specific datasheet for the delay parameters.

During the delays, the SVS will not flag a low-voltage condition or reset the device, and the SVSON bit is cleared. Software can test the SVSON bit to determine when the delay has elapsed and the SVS is monitoring the voltage properly.

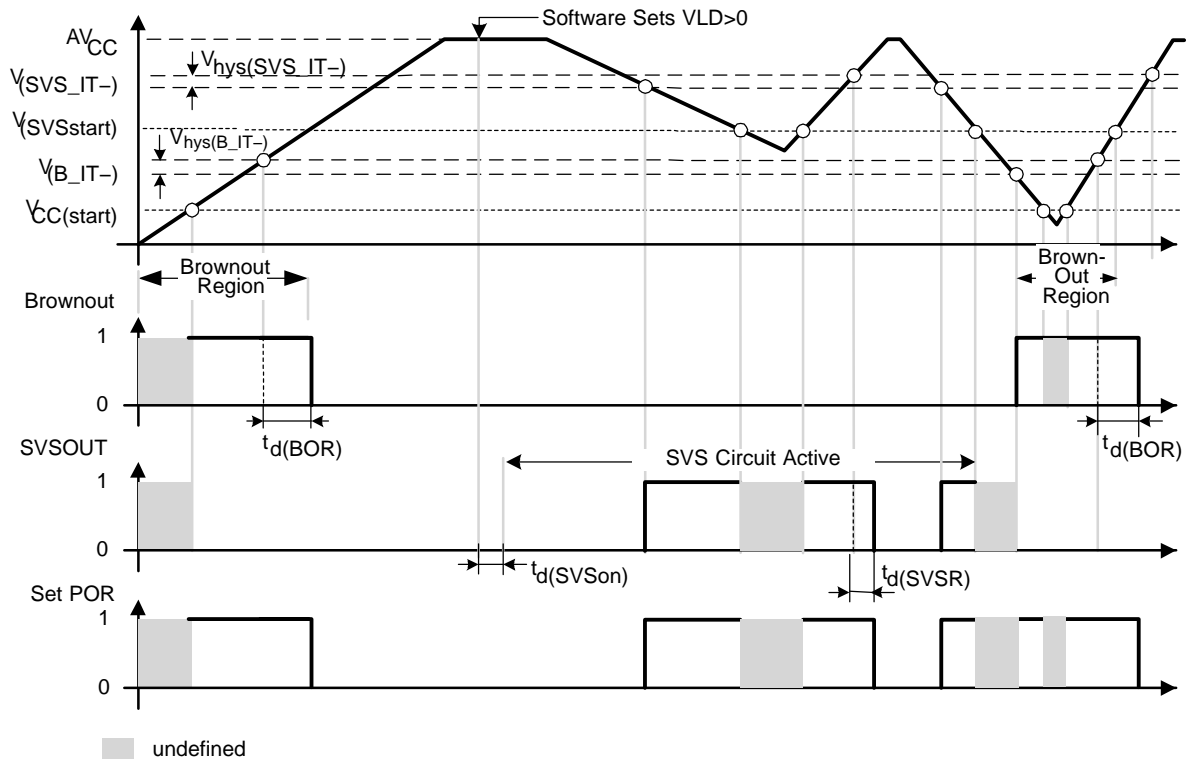
Figure 6–2. SVSON state When Changing VLDx



### 6.2.4 SVS Operating Range

Each SVS level has hysteresis to reduce sensitivity to small supply voltage changes when  $AV_{CC}$  is close to the threshold. The SVS operation and SVS/Brownout interoperation are shown in Figure 6–3.

Figure 6–3. Operating Levels for SVS and Brownout/Reset Circuit





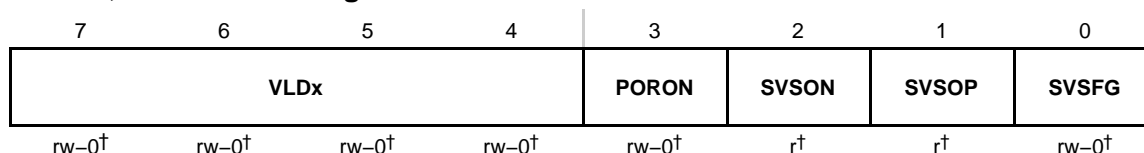
## 6.3 SVS Registers

The SVS registers are listed in Table 6–1.

Table 6–1. SVS Registers

Register	Short Form	Register Type	Address	Initial State
SVS Control Register	SVSCTL	Read/write	056h	Reset with BOR

### SVSCTL, SVS Control Register



† Reset by a brownout reset only, not by a POR or PUC.

<b>VLDx</b>	Bits 7-4	Voltage level detect. These bits turn on the SVS and select the nominal SVS threshold voltage level. See the device-specific datasheet for parameters. 0000 SVS is off 0001 1.9 V 0010 2.1 V 0011 2.2 V 0100 2.3 V 0101 2.4 V 0110 2.5 V 0111 2.65 V 1000 2.8 V 1001 2.9 V 1010 3.05 V 1011 3.2 V 1100 3.35 V 1101 3.5 V 1110 3.7 V 1111 Compares external input voltage SVSIN to 1.2 V.
<b>PORON</b>	Bit 3	POR on. This bit enables the SVSFG flag to cause a POR device reset. 0 SVSFG does not cause a POR 1 SVSFG causes a POR
<b>SVSON</b>	Bit 2	SVS on. This bit reflects the status of SVS operation. This bit DOES NOT turn on the SVS. The SVS is turned on by setting VLDx > 0. 0 SVS is Off 1 SVS is On
<b>SVSOP</b>	Bit 1	SVS output. This bit reflects the output value of the SVS comparator. 0 SVS comparator output is high 1 SVS comparator output is low
<b>SVSFG</b>	Bit 0	SVS flag. This bit indicates a low voltage condition. SVSFG remains set after a low voltage condition until reset by software. 0 No low voltage condition occurred 1 A low condition is present or has occurred

# Hardware Multiplier

---

---

---

---

This chapter describes the hardware multiplier. The hardware multiplier is implemented in MSP430x44x devices.

<b>Topic</b>	<b>Page</b>
7.1 Hardware Multiplier Introduction .....	7-2
7.2 Hardware Multiplier Operation .....	7-3
7.3 Hardware Multiplier Registers .....	7-7

## 7.1 Hardware Multiplier Introduction

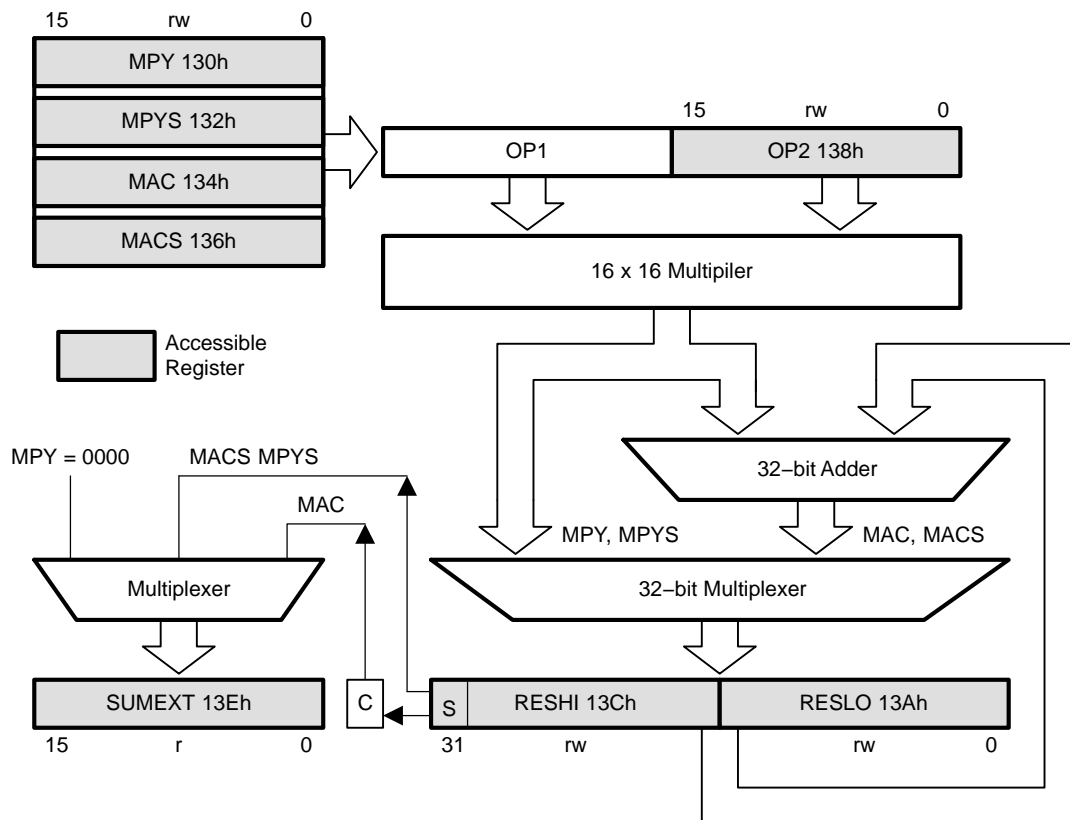
The hardware multiplier is a peripheral and is not part of the MSP430 CPU. This means, its activities do not interfere with the CPU activities. The multiplier registers are peripheral registers that are loaded and read with CPU instructions.

The hardware multiplier supports:

- Unsigned multiply
- Signed multiply
- Unsigned multiply accumulate
- Signed multiply accumulate
- 16×16 bits, 16×8 bits, 8×16 bits, 8×8 bits

The hardware multiplier block diagram is shown in Figure 7–1.

Figure 7–1. Hardware Multiplier Block Diagram



## 7.2 Hardware Multiplier Operation

The hardware multiplier supports unsigned multiply, signed multiply, unsigned multiply accumulate, and signed multiply accumulate operations. The type of operation is selected by the address the first operand is written to.

The hardware multiplier has two 16-bit operand registers, OP1 and OP2, and three result registers, RESLO, RESHI, and SUMEXT. RESLO stores the low word of the result, RESHI stores the high word of the result, and SUMEXT stores information about the result. The result is ready in three MCLK cycles and can be read with the next instruction after writing to OP2, except when using an indirect addressing mode to access the result. When using indirect addressing for the result, a NOP is required before the result is ready.

### 7.2.1 Operand Registers

The operand one register OP1 has four addresses, shown in Table 7–1, used to select the multiply mode. Writing the first operand to the desired address selects the type of multiply operation but does not start any operation. Writing the second operand to the operand two register OP2 initiates the multiply operation. Writing OP2 starts the selected operation with the values stored in OP1 and OP2. The result is written into the three result registers RESLO, RESHI, and SUMEXT.

Repeated multiply operations may be performed without reloading OP1 if the OP1 value is used for successive operations. It is not necessary to re-write the OP1 value to perform the operations.

Table 7–1. OP1 addresses

OP1 Address	Register Name	Operation
0130h	MPY	Unsigned multiply
0132h	MPYS	Signed multiply
0134h	MAC	Unsigned multiply accumulate
0136h	MACS	Signed multiply accumulate

## 7.2.2 Result Registers

The result low register RESLO holds the lower 16-bits of the calculation result. The result high register RESHI contents depend on the multiply operation and are listed in Table 7–2.

Table 7–2. RESHI Contents

Mode	RESHI Contents
MPY	Upper 16-bits of the result
MPYS	The MSB is the sign of the result. The remaining bits are the upper 15-bits of the result. Two's complement notation is used for the result.
MAC	Upper 16-bits of the result
MACS	Upper 16-bits of the result. Two's complement notation is used for the result.

The sum extension registers SUMEXT contents depend on the multiply operation and are listed in Table 7–3.

Table 7–3. SUMEXT Contents

Mode	SUMEXT
MPY	SUMEXT is always 0000h
MPYS	SUMEXT contains the extended sign of the result 00000h Result was positive or zero 0FFFFh Result was negative
MAC	SUMEXT contains the carry of the result 0000h No carry for result 0001h Result has a carry
MACS	SUMEXT contains the extended sign of the result 00000h Result was positive or zero 0FFFFh Result was negative

### MACS Underflow and Overflow

The multiplier does not automatically detect underflow or overflow in the MACS mode. The accumulator range for positive numbers is 0 to 7FFF FFFFh and for negative numbers is 0FFFF FFFFh to 8000 0000h. An overflow occurs when the sum of two negative numbers yields a result that is in the range for a positive number. An underflow occurs when the sum of two positive numbers yields a result that is in the range for a negative number. In both of these cases, the SUMEXT register contains the correct sign of the result, 0FFFFh for overflow and 0000h for underflow. User software must detect and handle these conditions appropriately.

### 7.2.3 Software Examples

Examples for all multiplier modes follow. All 8x8 modes use the absolute address for the registers because the assembler will not allow .B access to word registers when using the labels from the standard definitions file.

```

; 16x16 Unsigned Multiply
    MOV    #01234h,&MPY ; Load first operand
    MOV    #05678h,&OP2 ; Load second operand
; ...                ; Process results

; 8x8 Unsigned Multiply. Absolute addressing.
    MOV.B #012h,&0130h ; Load first operand
    MOV.B #034h,&0138h ; Load 2nd operand
; ...                ; Process results

; 16x16 Signed Multiply
    MOV    #01234h,&MPYS ; Load first operand
    MOV    #05678h,&OP2 ; Load 2nd operand
; ...                ; Process results

; 8x8 Signed Multiply. Absolute addressing.
    MOV.B #012h,&0132h ; Load first operand
    SXT    &MPYS        ; Sign extend first operand
    MOV.B #034h,&0138h ; Load 2nd operand
    SXT    &OP2         ; Sign extend 2nd operand
                        ; (triggers 2nd multiplication)
; ...                ; Process results

; 16x16 Unsigned Multiply Accumulate
    MOV    #01234h,&MAC ; Load first operand
    MOV    #05678h,&OP2 ; Load 2nd operand
; ...                ; Process results

; 8x8 Unsigned Multiply Accumulate. Absolute addressing
    MOV.B #012h,&0134h ; Load first operand
    MOV.B #034h,&0138h ; Load 2nd operand
; ...                ; Process results

; 16x16 Signed Multiply Accumulate
    MOV    #01234h,&MACS ; Load first operand
    MOV    #05678h,&OP2 ; Load 2nd operand
; ...                ; Process results

; 8x8 Signed Multiply Accumulate. Absolute addressing
    MOV.B #012h,&0136h ; Load first operand
    SXT    &MACS        ; Sign extend first operand
    MOV.B #034h,R5     ; Temp. location for 2nd operand
    SXT    R5           ; Sign extend 2nd operand
    MOV    R5,&OP2      ; Load 2nd operand
; ...                ; Process results

```

## 7.2.4 Indirect Addressing of RESLO

When using indirect or indirect autoincrement addressing mode to access the result registers, At least one instruction is needed between loading the second operand and accessing one of the result registers:

```
; Access multiplier results with indirect addressing
MOV  #RESLO,R5    ; RESLO address in R5 for indirect
MOV  &OPER1,&MPY  ; Load 1st operand
MOV  &OPER2,&OP2  ; Load 2nd operand
NOP                               ; Need one cycle
MOV  @R5+,&xxx    ; Move RESLO
MOV  @R5,&xxx     ; Move RESHI
```

## 7.2.5 Using Interrupts

If an interrupt occurs after writing OP1, but before writing OP2, and the multiplier is used in servicing that interrupt, the original multiplier mode selection is lost and the results are unpredictable. To avoid this, disable interrupts before using the hardware multiplier or do not use the multiplier in interrupt service routines.

```
; Disable interrupts before using the hardware multiplier
DINT                ; Disable interrupts
NOP                 ; Required for DINT
MOV  #xxh,&MPY      ; Load 1st operand
MOV  #xxh,&OP2      ; Load 2nd operand
EINT                ; Interrupts may be enable before
                   ; Process results
```

### 7.3 Hardware Multiplier Registers

The hardware multiplier registers are listed in Table 7–4.

*Table 7–4. Hardware Multiplier Registers*

<b>Register</b>	<b>Short Form</b>	<b>Register Type</b>	<b>Address</b>	<b>Initial State</b>
Operand one - multiply	MPY	Read/write	0130h	Unchanged
Operand one - signed multiply	MPYS	Read/write	0132h	Unchanged
Operand one - multiply accumulate	MAC	Read/write	0134h	Unchanged
Operand one - signed multiply accumulate	MACS	Read/write	0136h	Unchanged
Operand two	OP2	Read/write	0138h	Unchanged
Result low word	RESLO	Read/write	013Ah	Undefined
Result high word	RESHI	Read/write	013Ch	Undefined
Sum Extension register	SUMEXT	Read	013Eh	Undefined



# DMA Controller

---

---

---

---

The DMA controller module transfers data from one address to another without CPU intervention. This chapter describes the operation of the DMA controller. The DMA controller is implemented in MSP430FG43x and implements only one DMA channel.

<b>Topic</b>	<b>Page</b>
<b>8.1 DMA Introduction</b> .....	<b>8-2</b>
<b>8.2 DMA Operation</b> .....	<b>8-4</b>
<b>8.3 DMA Registers</b> .....	<b>8-18</b>

---

## 8.1 DMA Introduction

The direct memory access (DMA) controller transfers data from one address to another, without CPU intervention, across the entire address range. For example, the DMA controller can move data from the ADC12 conversion memory to RAM.

MSP430FG43x devices implement only one DMA channel. Therefore some features described in this chapter are not applicable to MSP430FG43x devices.

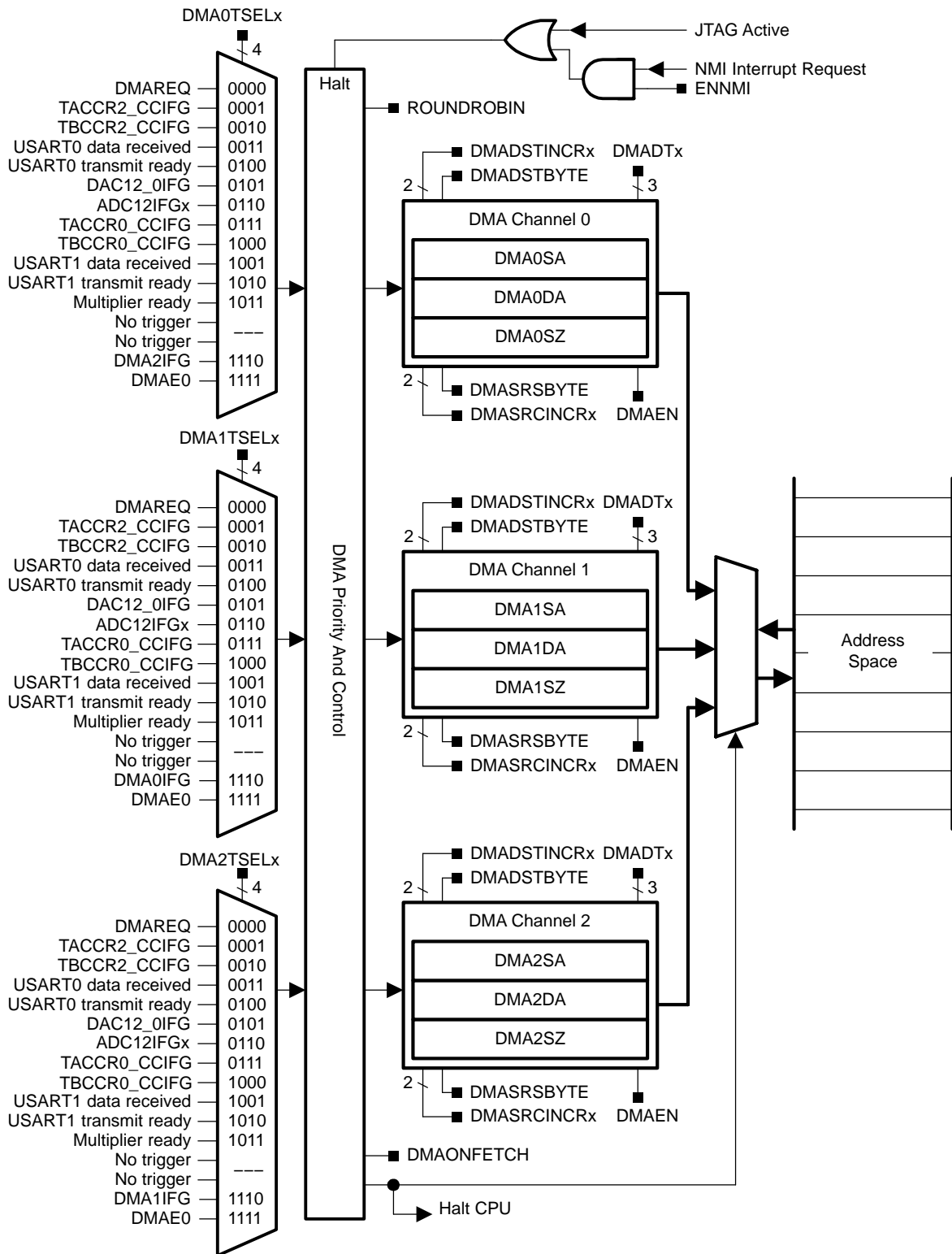
Using the DMA controller can increase the throughput of peripheral modules. It can also reduce system power consumption by allowing the CPU to remain in a low-power mode without having to awaken to move data to or from a peripheral.

The DMA controller features include:

- Up to three independent transfer channels
- Configurable DMA channel priorities
- Requires only two MCLK clock cycles
- Byte or word and mixed byte/word transfer capability
- Block sizes up to 65535 bytes or words
- Configurable transfer trigger selections
- Selectable edge or level-triggered transfer
- Four addressing modes
- Single, block, or burst-block transfer modes

The DMA controller block diagram is shown in Figure 8–1.

Figure 8-1. DMA Controller Block Diagram



## 8.2 DMA Operation

The DMA controller is configured with user software. The setup and operation of the DMA is discussed in the following sections.

### 8.2.1 DMA Addressing Modes

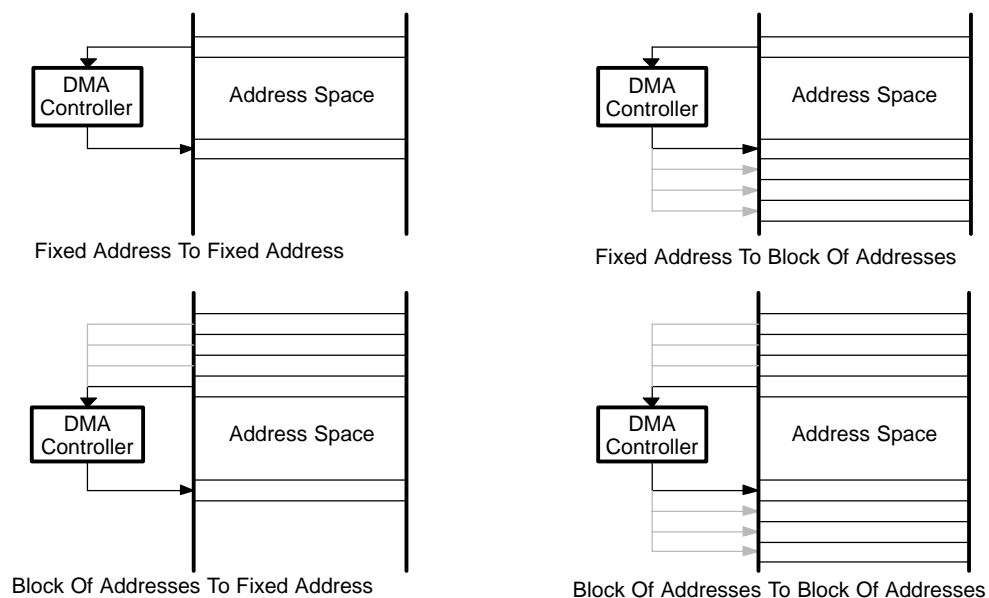
The DMA controller has four addressing modes. The addressing mode for each DMA channel is independently configurable. For example, channel 0 may transfer between two fixed addresses, while channel 1 transfers between two blocks of addresses. The addressing modes are shown in Figure 8–2. The addressing modes are:

- Fixed address to fixed address
- Fixed address to block of addresses
- Block of addresses to fixed address
- Block of addresses to block of addresses

The addressing modes are configured with the `DMASRCINCRx` and `DMADSTINCRx` control bits. The `DMASRCINCRx` bits select if the source address is incremented, decremented, or unchanged after each transfer. The `DMADSTINCRx` bits select if the destination address is incremented, decremented, or unchanged after each transfer.

Transfers may be byte-to-byte, word-to-word, byte-to-word, or word-to-byte. When transferring word-to-byte, only the lower byte of the source-word transfers. When transferring byte-to-word, the upper byte of the destination-word is cleared when the transfer occurs.

Figure 8–2. DMA Addressing Modes



---

## 8.2.2 DMA Transfer Modes

The DMA controller has six transfer modes selected by the DMADTx bits as listed in Table 8–1. Each channel is individually configurable for its transfer mode. For example, channel 0 may be configured in single transfer mode, while channel 1 is configured for burst-block transfer mode, and channel 2 operates in repeated block mode. The transfer mode is configured independently from the addressing mode. Any addressing mode can be used with any transfer mode.

Table 8–1. DMA Transfer Modes

DMADTx	Transfer Mode	Description
000	Single transfer	Each transfer requires a trigger. DMAEN is automatically cleared when DMAxSZ transfers have been made.
001	Block transfer	A complete block is transferred with one trigger. DMAEN is automatically cleared at the end of the block transfer.
010, 011	Burst-block transfer	CPU activity is interleaved with a block transfer. DMAEN is automatically cleared at the end of the burst-block transfer.
100	Repeated single transfer	Each transfer requires a trigger. DMAEN remains enabled.
101	Repeated block transfer	A complete block is transferred with one trigger. DMAEN remains enabled.
110, 111	Repeated burst-block transfer	CPU activity is interleaved with a block transfer. DMAEN remains enabled.

---

## Single Transfer

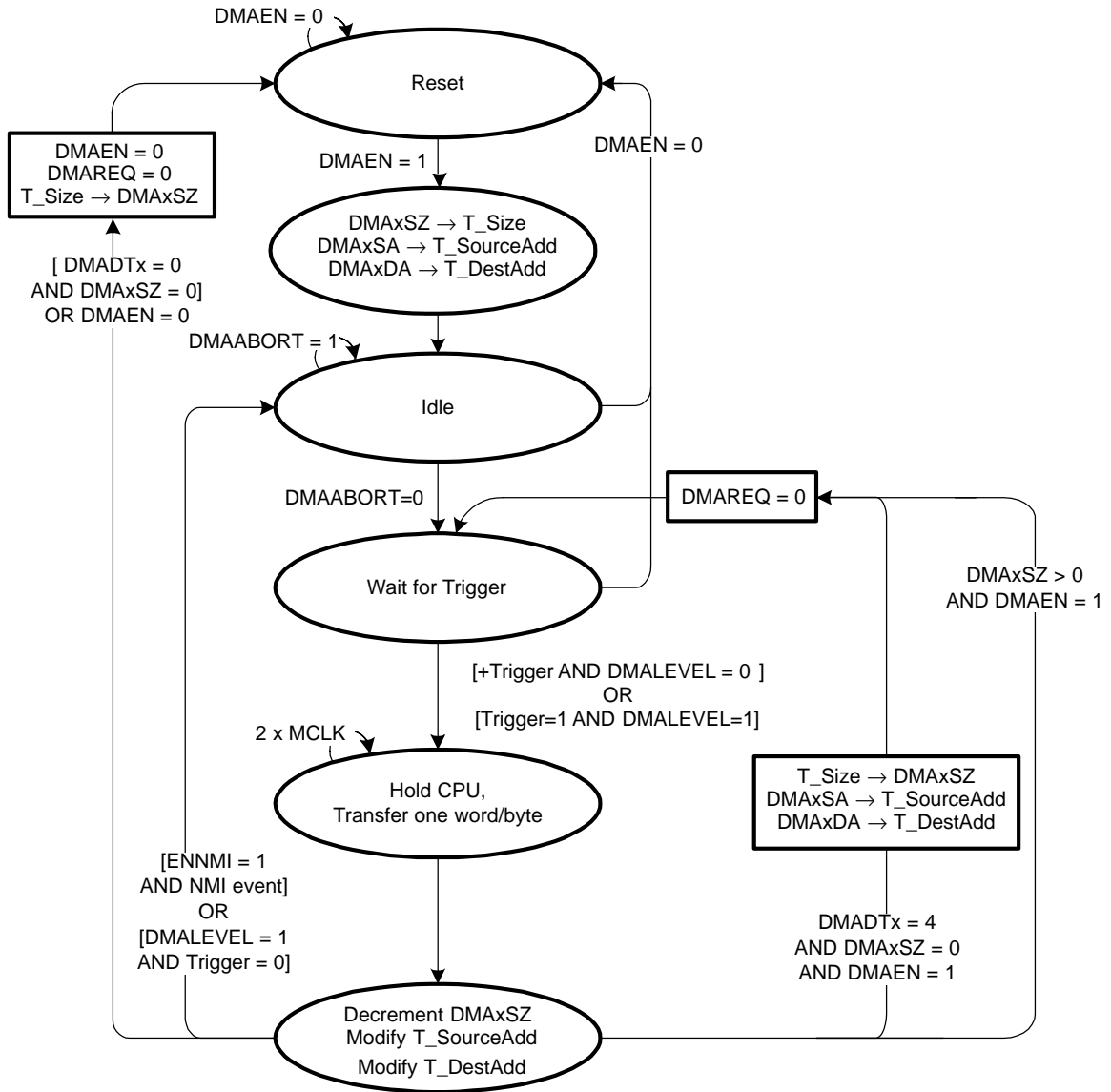
In single transfer mode, each byte/word transfer requires a separate trigger. The single transfer state diagram is shown in Figure 8–3.

The DMAxSZ register is used to define the number of transfers to be made. The DMADSTINCRx and DMASRCINCRx bits select if the destination address and the source address are incremented or decremented after each transfer. If DMAxSZ = 0, no transfers occur.

The DMAxSA, DMAxDA, and DMAxSZ registers are copied into temporary registers. The temporary values of DMAxSA and DMAxDA are incremented or decremented after each transfer. The DMAxSZ register is decremented after each transfer. When the DMAxSZ register decrements to zero it is reloaded from its temporary register and the corresponding DMAIFG flag is set. When DMADTx = 0, the DMAEN bit is cleared automatically when DMAxSZ decrements to zero and must be set again for another transfer to occur.

In repeated single transfer mode, the DMA controller remains enabled with DMAEN = 1, and a transfer occurs every time a trigger occurs.

Figure 8-3. DMA Single Transfer State Diagram



---

## Block Transfers

In block transfer mode, a transfer of a complete block of data occurs after one trigger. When  $DMADTx = 1$ , the  $DMAEN$  bit is cleared after the completion of the block transfer and must be set again before another block transfer can be triggered. After a block transfer has been triggered, further trigger signals occurring during the block transfer are ignored. The block transfer state diagram is shown in Figure 8–4.

The  $DMAxSZ$  register is used to define the size of the block and the  $DMADSTINCRx$  and  $DMASRCINCRx$  bits select if the destination address and the source address are incremented or decremented after each transfer of the block. If  $DMAxSZ = 0$ , no transfers occur.

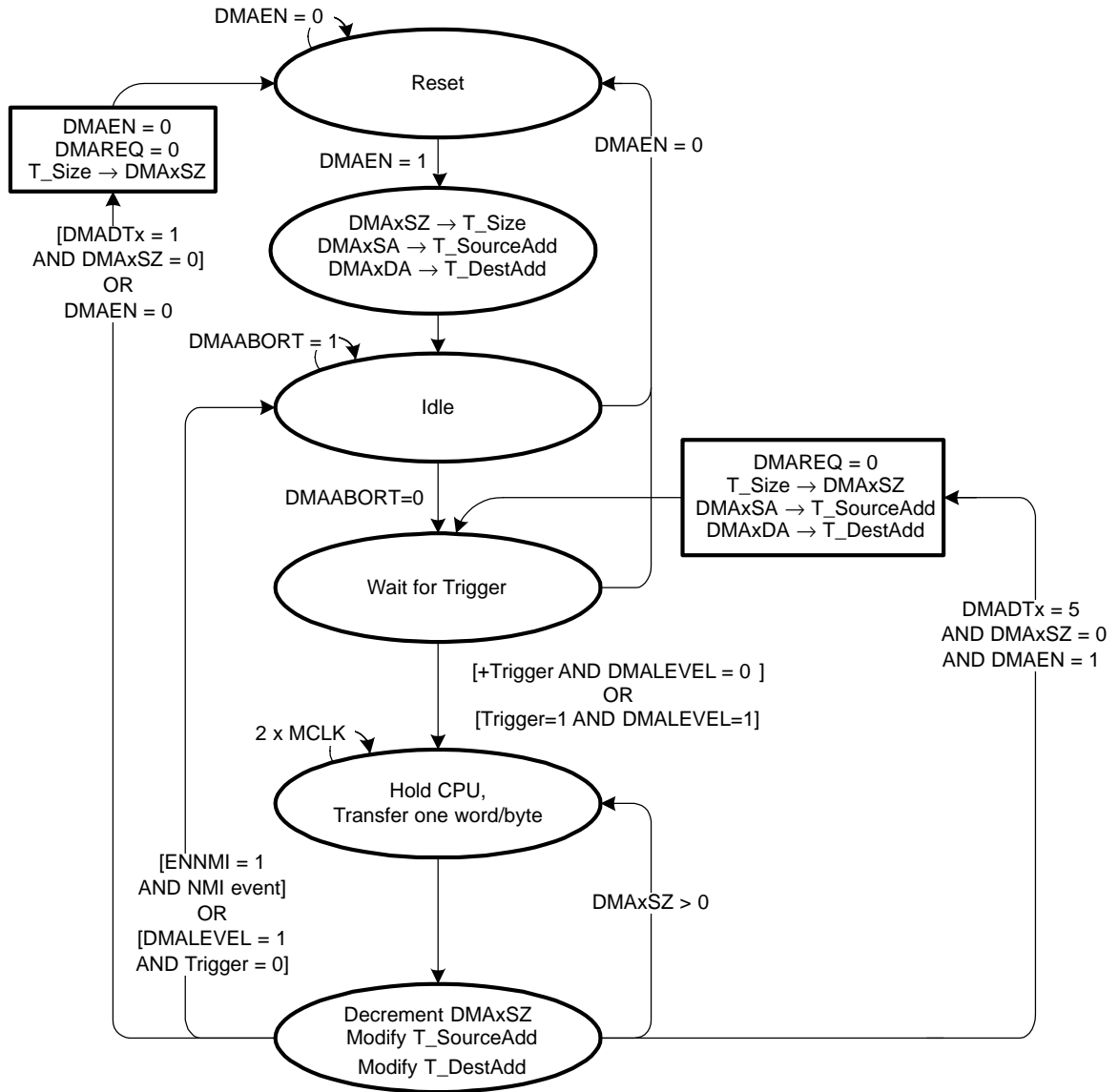
The  $DMAxSA$ ,  $DMAxDA$ , and  $DMAxSZ$  registers are copied into temporary registers. The temporary values of  $DMAxSA$  and  $DMAxDA$  are incremented or decremented after each transfer in the block. The  $DMAxSZ$  register is decremented after each transfer of the block and shows the number of transfers remaining in the block. When the  $DMAxSZ$  register decrements to zero it is reloaded from its temporary register and the corresponding  $DMAIFG$  flag is set.

During a block transfer, the CPU is halted until the complete block has been transferred. The block transfer takes  $2 \times MCLK \times DMAxSZ$  clock cycles to complete. CPU execution resumes with its previous state after the block transfer is complete.

In repeated block transfer mode, the  $DMAEN$  bit remains set after completion of the block transfer. The next trigger after the completion of a repeated block transfer triggers another block transfer.



Figure 8-4. DMA Block Transfer State Diagram



---

## Burst-Block Transfers

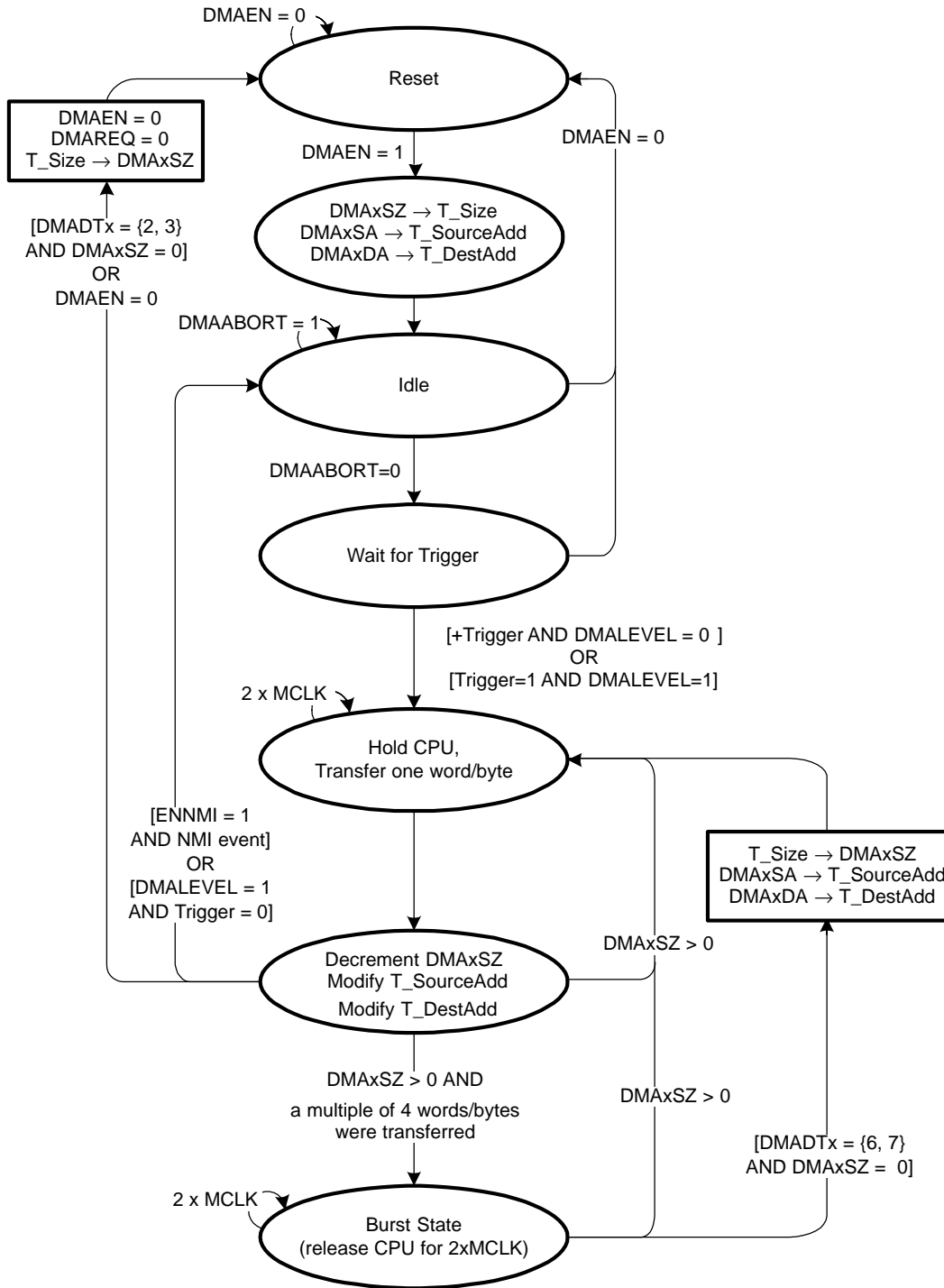
In burst-block mode, transfers are block transfers with CPU activity interleaved. The CPU executes 2 MCLK cycles after every four byte/word transfers of the block resulting in 20% CPU execution capacity. After the burst-block, CPU execution resumes at 100% capacity and the DMAEN bit is cleared. DMAEN must be set again before another burst-block transfer can be triggered. After a burst-block transfer has been triggered, further trigger signals occurring during the burst-block transfer are ignored. The burst-block transfer state diagram is shown in Figure 8–5.

The DMAxSZ register is used to define the size of the block and the DMADSTINCRx and DMASRCINCRx bits select if the destination address and the source address are incremented or decremented after each transfer of the block. If DMAxSZ = 0, no transfers occur.

The DMAxSA, DMAxDA, and DMAxSZ registers are copied into temporary registers. The temporary values of DMAxSA and DMAxDA are incremented or decremented after each transfer in the block. The DMAxSZ register is decremented after each transfer of the block and shows the number of transfers remaining in the block. When the DMAxSZ register decrements to zero it is reloaded from its temporary register and the corresponding DMAIFG flag is set.

In repeated burst-block mode the DMAEN bit remains set after completion of the burst-block transfer and no further trigger signals are required to initiate another burst-block transfer. Another burst-block transfer begins immediately after completion of a burst-block transfer. In this case, the transfers must be stopped by clearing the DMAEN bit, or by an NMI interrupt when ENNMI is set. In repeated burst-block mode the CPU executes at 20% capacity continuously until the repeated burst-block transfer is stopped.

Figure 8-5. DMA Burst-Block Transfer State Diagram



---

### 8.2.3 Initiating DMA Transfers

Each DMA channel is independently configured for its trigger source with the DMAxTSELx bits as described in Table 8–2. The DMAxTSELx bits should be modified only when the DMACTLx DMAEN bit is 0. Otherwise, unpredictable DMA triggers may occur.

When selecting the trigger, the trigger must not have already occurred, or the transfer will not take place. For example, if the TACCR2 CCIFG bit is selected as a trigger, and it is already set, no transfer will occur until the next time the TACCR2 CCIFG bit is set.

#### Edge-Sensitive Triggers

When DMALEVEL = 0, edge-sensitive triggers are used and the rising edge of the trigger signal initiates the transfer. In single-transfer mode, each transfer requires its own trigger. When using block or burst-block modes, only one trigger is required to initiate the block or burst-block transfer.

#### Level-Sensitive Triggers

When DMALEVEL = 1, level-sensitive triggers are used. For proper operation, level-sensitive triggers can only be used when external trigger DMAE0 is selected as the trigger. DMA transfers are triggered as long as the trigger signal is high and the DMAEN bit remains set.

The trigger signal must remain high for a block or burst-block transfer to complete. If the trigger signal goes low during a block or burst-block transfer, the DMA controller is held in its current state until the trigger goes back high or until the DMA registers are modified by software. If the DMA registers are not modified by software, when the trigger signal goes high again, the transfer resumes from where it was when the trigger signal went low.

When DMALEVEL = 1, transfer modes selected when DMADTx = {0, 1, 2, 3} are recommended because the DMAEN bit is automatically reset after the configured transfer.

#### Halting Executing Instructions for DMA Transfers

The DMAONFETCH bit controls when the CPU is halted for a DMA transfer. When DMAONFETCH = 0, the CPU is halted immediately and the transfer begins when a trigger is received. When DMAONFETCH = 1, the CPU finishes the currently executing instruction before the DMA controller halts the CPU and the transfer begins.

---

**Note: DMAONFETCH Must Be Used When The DMA Writes To Flash**

If the DMA controller is used to write to flash memory, the DMAONFETCH bit must be set. Otherwise, unpredictable operation can result.

---

Table 8–2. DMA Trigger Operation

DMAxTSELx	Operation
0000	A transfer is triggered when the DMAREQ bit is set. The DMAREQ bit is automatically reset when the transfer starts
0001	A transfer is triggered when the TACCR2 CCIFG flag is set. The TACCR2 CCIFG flag is automatically reset when the transfer starts. If the TACCR2 CCIE bit is set, the TACCR2 CCIFG flag will not trigger a transfer.
0010	A transfer is triggered when the TBCCR2 CCIFG flag is set. The TBCCR2 CCIFG flag is automatically reset when the transfer starts. If the TBCCR2 CCIE bit is set, the TBCCR2 CCIFG flag will not trigger a transfer.
0011	A transfer is triggered when USART0 receives new data. In I <sup>2</sup> C mode, the trigger is the data-received condition, not the RXRDYIFG flag. RXRDYIFG is not cleared when the transfer starts, and setting RXRDYIFG with software will not trigger a transfer. If RXRDYIE is set, the data received condition will not trigger a transfer. In UART or SPI mode, a transfer is triggered when the URXIFG0 flag is set. URXIFG0 is automatically reset when the transfer starts. If URXIE0 is set, the URXIFG0 flag will not trigger a transfer.
0100	A transfer is triggered when USART0 is ready to transmit new data. In I <sup>2</sup> C mode, the trigger is the transmit-ready condition, not the TXRDYIFG flag. TXRDYIFG is not cleared when the transfer starts, and setting TXRDYIFG with software will not trigger a transfer. If TXRDYIE is set, the transmit ready condition will not trigger a transfer. In UART or SPI mode, a transfer is triggered when the UTXIFG0 flag is set. UTXIFG0 is automatically reset when the transfer starts. If UTXIE0 is set, the UTXIFG0 flag will not trigger a transfer.
0101	A transfer is triggered when the DAC12_0CTL DAC12IFG flag is set. The DAC12_0CTL DAC12IFG flag is automatically cleared when the transfer starts. If the DAC12_0CTL DAC12IE bit is set, the DAC12_0CTL DAC12IFG flag will not trigger a transfer.
0110	A transfer is triggered by an ADC12IFGx flag. When single-channel conversions are performed, the corresponding ADC12IFGx is the trigger. When sequences are used, the ADC12IFGx for the last conversion in the sequence is the trigger. A transfer is triggered when the conversion is completed and the ADC12IFGx is set. Setting the ADC12IFGx with software will not trigger a transfer. All ADC12IFGx flags are automatically reset when the associated ADC12MEMx register is accessed by the DMA controller.
0111	A transfer is triggered when the TACCR0 CCIFG flag is set. The TACCR0 CCIFG flag is automatically reset when the transfer starts. If the TACCR0 CCIE bit is set, the TACCR0 CCIFG flag will not trigger a transfer.
1000	A transfer is triggered when the TBCCR0 CCIFG flag is set. The TBCCR0 CCIFG flag is automatically reset when the transfer starts. If the TBCCR0 CCIE bit is set, the TBCCR0 CCIFG flag will not trigger a transfer.
1001	A transfer is triggered when the URXIFG1 flag is set. URXIFG1 is automatically reset when the transfer starts. If URXIE1 is set, the URXIFG1 flag will not trigger a transfer.
1010	A transfer is triggered when the UTXIFG1 flag is set. UTXIFG1 is automatically reset when the transfer starts. If UTXIE1 is set, the UTXIFG1 flag will not trigger a transfer.
1011	A transfer is triggered when the hardware multiplier is ready for a new operand.
1100	No transfer is triggered.
1101	No transfer is triggered.
1110	A transfer is triggered when the DMAxIFG flag is set. DMA0IFG triggers channel 1, DMA1IFG triggers channel 2, and DMA2IFG triggers channel 0. None of the DMAxIFG flags are automatically reset when the transfer starts.
1111	A transfer is triggered by the external trigger DMAE0.

---

## 8.2.4 Stopping DMA Transfers

There are two ways to stop DMA transfers in progress:

- A single, block, or burst-block transfer may be stopped with an NMI interrupt, if the ENNMI bit is set in register DMACTL1.
- A burst-block transfer may be stopped by clearing the DMAEN bit.

## 8.2.5 DMA Channel Priorities

The default DMA channel priorities are DMA0–DMA1–DMA2. If two or three triggers happen simultaneously or are pending, the channel with the highest priority completes its transfer (single, block or burst-block transfer) first, then the second priority channel, then the third priority channel. Transfers in progress are not halted if a higher priority channel is triggered. The higher priority channel waits until the transfer in progress completes before starting.

The DMA channel priorities are configurable with the ROUNDROBIN bit. When the ROUNDROBIN bit is set, the channel that completes a transfer becomes the lowest priority. The *order* of the priority of the channels always stays the same, DMA0–DMA1–DMA2, for example:

<b>DMA Priority</b>	<b>Transfer Occurs</b>	<b>New DMA Priority</b>
DMA0 – DMA1 – DMA2	DMA1	DMA2 – DMA0 – DMA1
DMA2 – DMA0 – DMA1	DMA2	DMA0 – DMA1 – DMA2
DMA0 – DMA1 – DMA2	DMA0	DMA1 – DMA2 – DMA0

When the ROUNDROBIN bit is cleared the channel priority returns to the default priority.

DMA channel priorities are not applicable to MSP430FG43x devices.

## 8.2.6 DMA Transfer Cycle Time

The DMA controller requires one or two MCLK clock cycles to synchronize before each single transfer or complete block or burst-block transfer. Each byte/word transfer requires two MCLK cycles after synchronization, and one cycle of wait time after the transfer. Because the DMA controller uses MCLK, the DMA cycle time is dependent on the MSP430 operating mode and clock system setup.

If the MCLK source is active, but the CPU is off, the DMA controller will use the MCLK source for each transfer, without re-enabling the CPU. If the MCLK source is off, the DMA controller will temporarily restart MCLK, sourced with DCOCLK, for the single transfer or complete block or burst-block transfer. The CPU remains off, and after the transfer completes, MCLK is turned off. The maximum DMA cycle time for all operating modes is shown in Table 8–3.

Table 8–3. Maximum Single-Transfer DMA Cycle Time

CPU Operating Mode	Clock Source	Maximum DMA Cycle Time
Active mode	MCLK=DCOCLK	4 MCLK cycles
Active mode	MCLK=LFXT1CLK	4 MCLK cycles
Low-power mode LPM0/1	MCLK=DCOCLK	5 MCLK cycles
Low-power mode LPM3/4	MCLK=DCOCLK	5 MCLK cycles + 6 $\mu$ s <sup>†</sup>
Low-power mode LPM0/1	MCLK=LFXT1CLK	5 MCLK cycles
Low-power mode LPM3	MCLK=LFXT1CLK	5 MCLK cycles
Low-power mode LPM4	MCLK=LFXT1CLK	5 MCLK cycles + 6 $\mu$ s <sup>†</sup>

<sup>†</sup> The additional 6  $\mu$ s are needed to start the DCOCLK. It is the  $t_{(LPMx)}$  parameter in the data sheet.

---

## 8.2.7 Using DMA with System Interrupts

DMA transfers are not interruptible by system interrupts. System interrupts remain pending until the completion of the transfer. NMI interrupts can interrupt the DMA controller if the ENNMI bit is set.

System interrupt service routines are interrupted by DMA transfers. If an interrupt service routine or other routine must execute with no interruptions, the DMA controller should be disabled prior to executing the routine.

## 8.2.8 DMA Controller Interrupts

Each DMA channel has its own DMAIFG flag. Each DMAIFG flag is set in any mode, when the corresponding DMAxSZ register counts to zero. If the corresponding DMAIE and GIE bits are set, an interrupt request is generated.

All DMAIFG flags source only one DMA controller interrupt vector and the interrupt vector is shared with the DAC12 module. Software must check the DMAIFG and DAC12IFG flags to determine the source of the interrupt. The DMAIFG flags are not reset automatically and must be reset by software.



---

## 8.2.9 Using the I<sup>2</sup>C Module with the DMA Controller

The I<sup>2</sup>C module provides two trigger sources for the DMA controller. The I<sup>2</sup>C module can trigger a transfer when new I<sup>2</sup>C data is received and the when the transmit data is needed.

The TXDMAEN and RXDMAEN bits enable or disable the use of the DMA controller with the I<sup>2</sup>C module. When RXDMAEN = 1, the DMA controller can be used to transfer data from the I<sup>2</sup>C module after the I<sup>2</sup>C modules receives data. When RXDMAEN = 1, RXRDYIE is ignored and RXRDYIFG will not generate an interrupt.

When TXDMAEN = 1, the DMA controller can be used to transfer data to the I<sup>2</sup>C module for transmission. When TXDMAEN = 1, TXRDYIE is ignored and TXRDYIFG will not generate an interrupt.

## 8.2.10 Using ADC12 with the DMA Controller

MSP430 devices with an integrated DMA controller can automatically move data from any ADC12MEMx register to another location. DMA transfers are done without CPU intervention and independently of any low-power modes. The DMA controller increases throughput of the ADC12 module, and enhances low-power applications allowing the CPU to remain off while data transfers occur.

DMA transfers can be triggered from any ADC12IFGx flag. When CONSEQx = {0,2} the ADC12IFGx flag for the ADC12MEMx used for the conversion can trigger a DMA transfer. When CONSEQx = {1,3}, the ADC12IFGx flag for the last ADC12MEMx in the sequence can trigger a DMA transfer. Any ADC12IFGx flag is automatically cleared when the DMA controller accesses the corresponding ADC12MEMx.

## 8.2.11 Using DAC12 With the DMA Controller

MSP430 devices with an integrated DMA controller can automatically move data to the DAC12\_xDAT register. DMA transfers are done without CPU intervention and independently of any low-power modes. The DMA controller increases throughput to the DAC12 module, and enhances low-power applications allowing the CPU to remain off while data transfers occur.

Applications requiring periodic waveform generation can benefit from using the DMA controller with the DAC12. For example, an application that produces a sinusoidal waveform may store the sinusoid values in a table. The DMA controller can continuously and automatically transfer the values to the DAC12 at specific intervals creating the sinusoid with zero CPU execution. The DAC12\_xCTL DAC12IFG flag is automatically cleared when the DMA controller accesses the DAC12\_xDAT register.

---

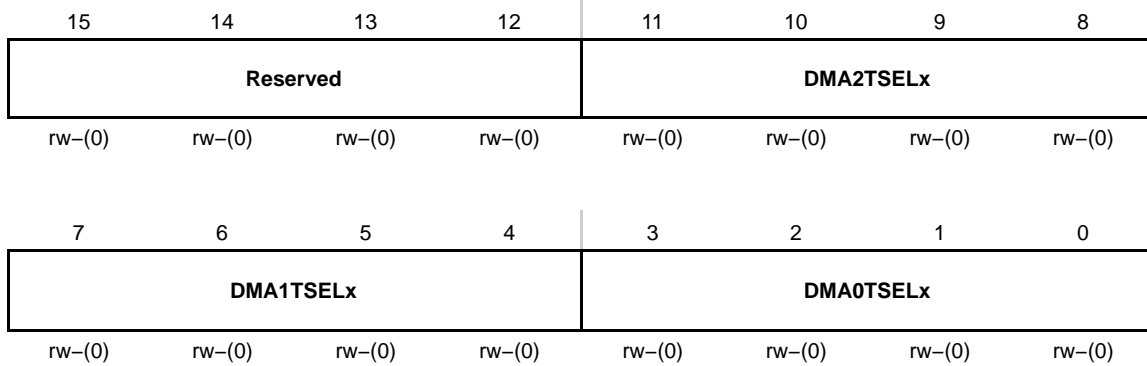
## 8.3 DMA Registers

The DMA registers are listed in Table 8–4.

*Table 8–4. DMA Registers*

<b>Register</b>	<b>Short Form</b>	<b>Register Type</b>	<b>Address</b>	<b>Initial State</b>
DMA control 0	DMACTL0	Read/write	0122h	Reset with POR
DMA control 1	DMACTL1	Read/write	0124h	Reset with POR
DMA channel 0 control	DMA0CTL	Read/write	01E0h	Reset with POR
DMA channel 0 source address	DMA0SA	Read/write	01E2h	Unchanged
DMA channel 0 destination address	DMA0DA	Read/write	01E4h	Unchanged
DMA channel 0 transfer size	DMA0SZ	Read/write	01E6h	Unchanged
DMA channel 1 control	DMA1CTL	Read/write	01E8h	Reset with POR
DMA channel 1 source address	DMA1SA	Read/write	01EAh	Unchanged
DMA channel 1 destination address	DMA1DA	Read/write	01ECh	Unchanged
DMA channel 1 transfer size	DMA1SZ	Read/write	01EEh	Unchanged
DMA channel 2 control	DMA2CTL	Read/write	01F0h	Reset with POR
DMA channel 2 source address	DMA2SA	Read/write	01F2h	Unchanged
DMA channel 2 destination address	DMA2DA	Read/write	01F4h	Unchanged
DMA channel 2 transfer size	DMA2SZ	Read/write	01F6h	Unchanged

## DMACTL0, DMA Control Register 0



<b>Reserved</b>	Bits 15–12	Reserved
<b>DMA2 TSELx</b>	Bits 11–8	DMA trigger select. These bits select the DMA transfer trigger. 0000 DMAREQ bit (software trigger) 0001 TACCR2 CCIFG bit 0010 TBCCR2 CCIFG bit 0011 URXIFG0 (UART/SPI mode), USART0 data received (I <sup>2</sup> C mode) 0100 UTXIFG0 (UART/SPI mode), USART0 transmit ready (I <sup>2</sup> C mode) 0101 DAC12_OCTL DAC12IFGx bit 0110 ADC12 ADC12IFGx bit 0111 TACCR0 CCIFG bit 1000 TBCCR0 CCIFG bit 1001 URXIFG1 bit 1010 UTXIFG1 bit 1011 Multiplier ready 1100 No action 1101 No action 1110 DMA0IFG bit triggers DMA channel 1 DMA1IFG bit triggers DMA channel 2 DMA2IFG bit triggers DMA channel 0 1111 External trigger DMAE0
<b>DMA1 TSELx</b>	Bits 7–4	Same as DMA2TSELx
<b>DMA0 TSELx</b>	Bits 3–0	Same as DMA2TSELx

## DMACTL1, DMA Control Register 1

15	14	13	12	11	10	9	8
0	0	0	0	0	0	0	0
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
0	0	0	0	0	DMA ONFETCH	ROUND ROBIN	ENNMI
r0	r0	r0	r0	r0	rw-(0)	rw-(0)	rw-(0)

<b>Reserved</b>	Bits 15-3	Reserved. Read only. Always read as 0.
<b>DMA ONFETCH</b>	Bit 2	DMA on fetch 0 The DMA transfer occurs immediately 1 The DMA transfer occurs on next instruction fetch after the trigger
<b>ROUND ROBIN</b>	Bit 1	Round robin. This bit enables the round-robin DMA channel priorities. 0 DMA channel priority is DMA0 – DMA1 – DMA2 1 DMA channel priority changes with each transfer
<b>ENNMI</b>	Bit 0	Enable NMI. This bit enables the interruption of a DMA transfer by an NMI interrupt. When an NMI interrupts a DMA transfer, the current transfer is completed normally, further transfers are stopped, and DMAABORT is set. 0 NMI interrupt does not interrupt DMA transfer 1 NMI interrupt interrupts a DMA transfer

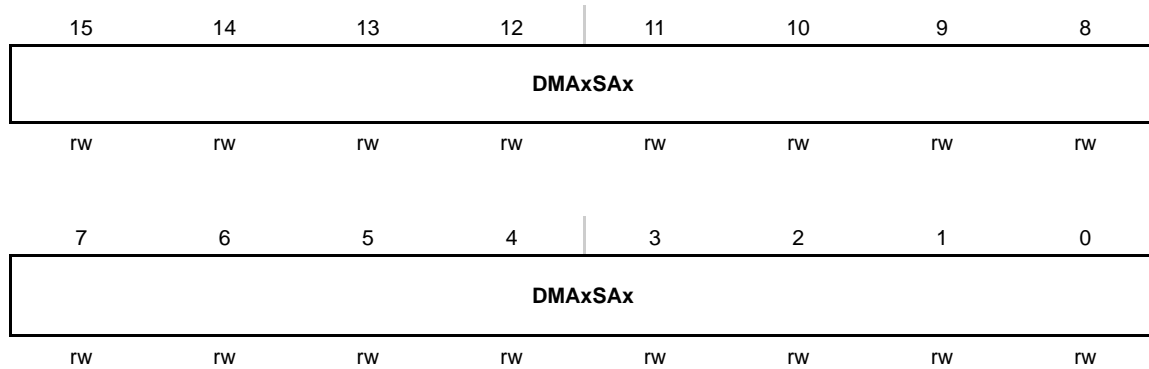
## DMAxCTL, DMA Channel x Control Register

15	14	13	12	11	10	9	8
<b>Reserved</b>	<b>DMADTx</b>			<b>DMADSTINCRx</b>		<b>DMASRCINCRx</b>	
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
<b>DMA DSTBYTE</b>	<b>DMA SRCBYTE</b>	<b>DMALEVEL</b>	<b>DMAEN</b>	<b>DMAIFG</b>	<b>DMAIE</b>	<b>DMA ABORT</b>	<b>DMAREQ</b>
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

<b>Reserved</b>	Bit 15	Reserved
<b>DMADTx</b>	Bits 14–12	<p>DMA Transfer mode.</p> <p>000 Single transfer</p> <p>001 Block transfer</p> <p>010 Burst-block transfer</p> <p>011 Burst-block transfer</p> <p>100 Repeated single transfer</p> <p>101 Repeated block transfer</p> <p>110 Repeated burst-block transfer</p> <p>111 Repeated burst-block transfer</p>
<b>DMA DSTINCRx</b>	Bits 11–10	<p>DMA destination increment. This bit selects automatic incrementing or decrementing of the destination address after each byte or word transfer. When DMADSTBYTE=1, the destination address increments/decrements by one. When DMADSTBYTE=0, the destination address increments/decrements by two. The DMAxDA is copied into a temporary register and the temporary register is incremented or decremented. DMAxDA is not incremented or decremented.</p> <p>00 Destination address is unchanged</p> <p>01 Destination address is unchanged</p> <p>10 Destination address is decremented</p> <p>11 Destination address is incremented</p>
<b>DMA SRCINCRx</b>	Bits 9–8	<p>DMA source increment. This bit selects automatic incrementing or decrementing of the source address for each byte or word transfer. When DMASRCBYTE=1, the source address increments/decrements by one. When DMASRCBYTE=0, the source address increments/decrements by two. The DMAxSA is copied into a temporary register and the temporary register is incremented or decremented. DMAxSA is not incremented or decremented.</p> <p>00 Source address is unchanged</p> <p>01 Source address is unchanged</p> <p>10 Source address is decremented</p> <p>11 Source address is incremented</p>
<b>DMA DSTBYTE</b>	Bit 7	<p>DMA destination byte. This bit selects the destination as a byte or word.</p> <p>0 Word</p> <p>1 Byte</p>

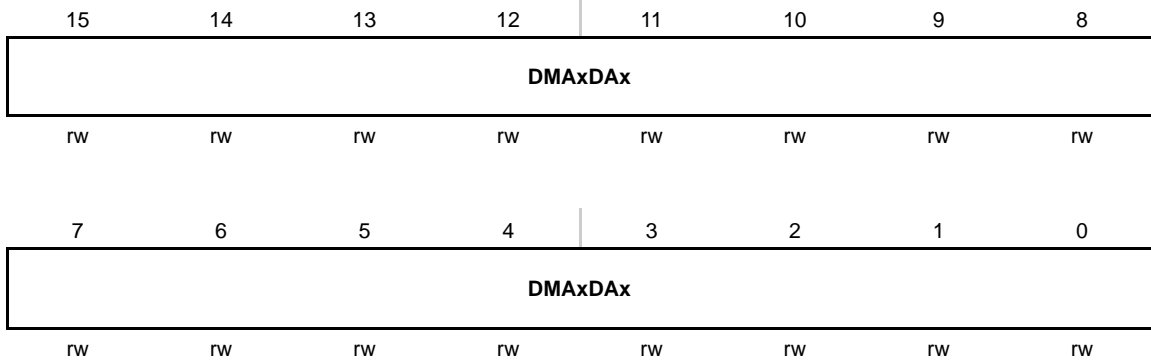
<b>DMA SRCBYTE</b>	Bit 6	DMA source byte. This bit selects the source as a byte or word. 0 Word 1 Byte
<b>DMA LEVEL</b>	Bit 5	DMA level. This bit selects between edge-sensitive and level-sensitive triggers. 0 Edge sensitive (rising edge) 1 Level sensitive (high level)
<b>DMAEN</b>	Bit 4	DMA enable 0 Disabled 1 Enabled
<b>DMAIFG</b>	Bit 3	DMA interrupt flag 0 No interrupt pending 1 Interrupt pending
<b>DMAIE</b>	Bit 2	DMA interrupt enable 0 Disabled 1 Enabled
<b>DMA ABORT</b>	Bit 1	DMA Abort. This bit indicates if a DMA transfer was interrupt by an NMI. 0 DMA transfer not interrupted 1 DMA transfer was interrupted by NMI
<b>DMAREQ</b>	Bit 0	DMA request. Software-controlled DMA start. DMAREQ is reset automatically. 0 No DMA start 1 Start DMA

### DMAxSA, DMA Source Address Register



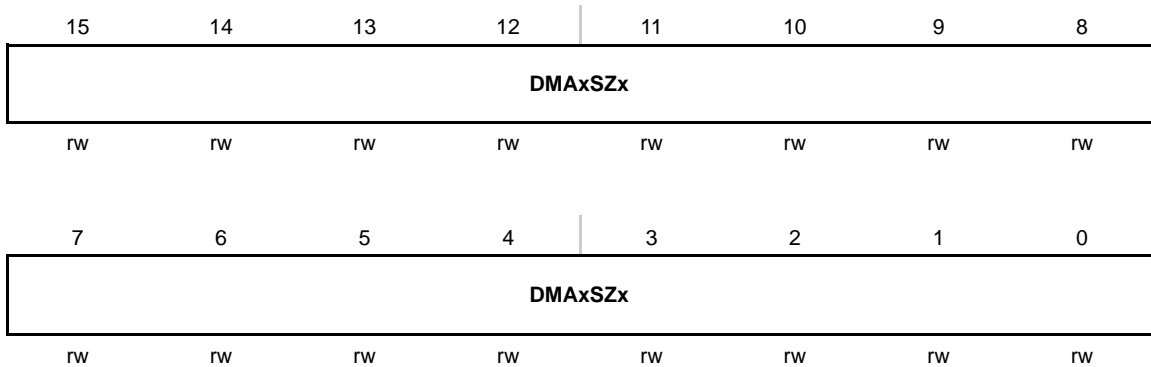
**DMAxSAx** Bits 15–0 DMA source address. The source address register points to the DMA source address for single transfers or the first source address for block transfers. The source address register remains unchanged during block and burst-block transfers.

## DMAxDA, DMA Destination Address Register



**DMAxDAx** Bits 15–0 DMA destination address. The destination address register points to the destination address for single transfers or the first address for block transfers. The DMAxDA register remains unchanged during block and burst-block transfers.

## DMAxSZ, DMA Size Address Register



**DMAxSZx** Bits 15–0 DMA size. The DMA size register defines the number of byte/word data per block transfer. DMAxSZ register decrements with each word or byte transfer. When DMAxSZ decrements to 0, it is immediately and automatically reloaded with its previously initialized value.

- 00000h Transfer is disabled
- 00001h One byte or word is transferred
- 00002h Two bytes or words are transferred
- :
- 0FFFFh 65535 bytes or words are transferred

# **Digital I/O**

---

---

---

---

This chapter describes the operation of the digital I/O ports. Ports P1-P6 are implemented in all MSP430x4xx devices.

<b>Topic</b>	<b>Page</b>
<b>9.1 Digital I/O Introduction</b> .....	<b>9-2</b>
<b>9.2 Digital I/O Operation</b> .....	<b>9-3</b>
<b>9.3 Digital I/O Registers</b> .....	<b>9-7</b>



## 9.1 Digital I/O Introduction

MSP430 devices have up to 6 digital I/O ports implemented, P1 - P6. Each port has eight I/O pins. Every I/O pin is individually configurable for input or output direction, and each I/O line can be individually read or written to.

Ports P1 and P2 have interrupt capability. Each interrupt for the P1 and P2 I/O lines can be individually enabled and configured to provide an interrupt on a rising edge or falling edge of an input signal. All P1 I/O lines source a single interrupt vector, and all P2 I/O lines source a different, single interrupt vector.

The digital I/O features include:

- Independently programmable individual I/Os
- Any combination of input or output
- Individually configurable P1 and P2 interrupts
- Independent input and output data registers

## 9.2 Digital I/O Operation

The digital I/O is configured with user software. The setup and operation of the digital I/O is discussed in the following sections.

### 9.2.1 Input Register PxIN

Each bit in each PxIN register reflects the value of the input signal at the corresponding I/O pin when the pin is configured as I/O function.

Bit = 0: The input is low

Bit = 1: The input is high

**Note: Writing to Read-Only Registers PxIN**

Writing to these read-only registers results in increased current consumption while the write attempt is active.

### 9.2.2 Output Registers PxOUT

Each bit in each PxOUT register is the value to be output on the corresponding I/O pin when the pin is configured as I/O function and output direction.

Bit = 0: The output is low

Bit = 1: The output is high

### 9.2.3 Direction Registers PxDIR

Each bit in each PxDIR register selects the direction of the corresponding I/O pin, regardless of the selected function for the pin. PxDIR bits for I/O pins that are selected for other module functions must be set as required by the other function.

Bit = 0: The port pin is switched to input direction

Bit = 1: The port pin is switched to output direction

## 9.2.4 Function Select Registers PxSEL

Port pins are often multiplexed with other peripheral module functions. See the device-specific data sheet to determine pin functions. Each PxSEL bit is used to select the pin function – I/O port or peripheral module function.

Bit = 0: I/O Function is selected for the pin

Bit = 1: Peripheral module function is selected for the pin

Setting PxSELx = 1 does not automatically set the pin direction. Other peripheral module functions may require the PxDIRx bits to be configured according to the direction needed for the module function. See the pin schematics in the device-specific datasheet.

```
;Output ACLK on P1.5 on MSP430F41x
BIS.B #020h,&P1SEL ; Select ACLK function for pin
BIS.B #020h,&P1DIR ; Set direction to output *Required*
```

**Note: P1 and P2 Interrupts Are Disabled When PxSEL = 1**

When any P1SELx or P2SELx bit is set, the corresponding pin's interrupt function is disabled. Therefore, signals on these pins will not generate P1 or P2 interrupts, regardless of the state of the corresponding P1IE or P2IE bit.

When a port pin is selected as an input to a peripheral, the input signal to the peripheral is a latched representation of the signal at the device pin. While PxSELx=1, the internal input signal follows the signal at the pin. However, if the PxSELx=0, the input to the peripheral maintains the value of the input signal at the device pin before the PxSELx bit was reset.

## 9.2.5 P1 and P2 Interrupts

Each pin in ports P1 and P2 have interrupt capability, configured with the PxIFG, PxIE, and PxIES registers. All P1 pins source a single interrupt vector, and all P2 pins source a different single interrupt vector. The PxIFG register can be tested to determine the source of a P1 or P2 interrupt.

### Interrupt Flag Registers P1IFG, P2IFG

Each PxIFGx bit is the interrupt flag for its corresponding I/O pin and is set when the selected input signal edge occurs at the pin. All PxIFGx interrupt flags request an interrupt when their corresponding PxIE bit and the GIE bit are set. Each PxIFG flag must be reset with software. Software can also set each PxIFG flag, providing a way to generate a software initiated interrupt.

Bit = 0: No interrupt is pending

Bit = 1: An interrupt is pending

Only transitions, not static levels, cause interrupts. If any PxIFGx flag becomes set during a Px interrupt service routine, or is set after the RETI instruction of a Px interrupt service routine is executed, the set PxIFGx flag generates another interrupt. This ensures that each transition is acknowledged.

**Note: PxIFG Flags When Changing PxOUT or PxDIR**

Writing to P1OUT, P1DIR, P2OUT, or P2DIR can result in setting the corresponding P1IFG or P2IFG flags.

**Note: Length of I/O Pin Interrupt Event**

Any external interrupt event should be at least 1.5 times MCLK or longer, to ensure that it is accepted and the corresponding interrupt flag is set.

## Interrupt Edge Select Registers P1IES, P2IES

Each PxIES bit selects the interrupt edge for the corresponding I/O pin.

Bit = 0: The PxIFGx flag is set with a low-to-high transition

Bit = 1: The PxIFGx flag is set with a high-to-low transition

---

### Note: Writing to PxIESx

Writing to P1IES, or P2IES can result in setting the corresponding interrupt flags.

PxIESx	PxINx	PxIFGx
0 → 1	0	May be set
0 → 1	1	Unchanged
1 → 0	0	Unchanged
1 → 0	1	May be set

## Interrupt Enable P1IE, P2IE

Each PxIE bit enables the associated PxIFG interrupt flag.

Bit = 0: The interrupt is disabled

Bit = 1: The interrupt is enabled

## 9.2.6 Configuring Unused Port Pins

Unused I/O pins should be configured as I/O function, output direction, and left unconnected on the PC board, to reduce power consumption. The value of the PxOUT bit is don't care, since the pin is unconnected. See chapter *System Resets, Interrupts, and Operating Modes* for termination unused pins.

### 9.3 Digital I/O Registers

Seven registers are used to configure P1 and P2. Four registers are used to configure ports P3 - P6. The digital I/O registers are listed in Table 9–1.

Table 9–1. Digital I/O Registers

Port	Register	Short Form	Address	Register Type	Initial State
P1	Input	P1IN	020h	Read only	–
	Output	P1OUT	021h	Read/write	Unchanged
	Direction	P1DIR	022h	Read/write	Reset with PUC
	Interrupt Flag	P1IFG	023h	Read/write	Reset with PUC
	Interrupt Edge Select	P1IES	024h	Read/write	Unchanged
	Interrupt Enable	P1IE	025h	Read/write	Reset with PUC
	Port Select	P1SEL	026h	Read/write	Reset with PUC
P2	Input	P2IN	028h	Read only	–
	Output	P2OUT	029h	Read/write	Unchanged
	Direction	P2DIR	02Ah	Read/write	Reset with PUC
	Interrupt Flag	P2IFG	02Bh	Read/write	Reset with PUC
	Interrupt Edge Select	P2IES	02Ch	Read/write	Unchanged
	Interrupt Enable	P2IE	02Dh	Read/write	Reset with PUC
	Port Select	P2SEL	02Eh	Read/write	Reset with PUC
P3	Input	P3IN	018h	Read only	–
	Output	P3OUT	019h	Read/write	Unchanged
	Direction	P3DIR	01Ah	Read/write	Reset with PUC
	Port Select	P3SEL	01Bh	Read/write	Reset with PUC
P4	Input	P4IN	01Ch	Read only	–
	Output	P4OUT	01Dh	Read/write	Unchanged
	Direction	P4DIR	01Eh	Read/write	Reset with PUC
	Port Select	P4SEL	01Fh	Read/write	Reset with PUC
P5	Input	P5IN	030h	Read only	–
	Output	P5OUT	031h	Read/write	Unchanged
	Direction	P5DIR	032h	Read/write	Reset with PUC
	Port Select	P5SEL	033h	Read/write	Reset with PUC
P6	Input	P6IN	034h	Read only	–
	Output	P6OUT	035h	Read/write	Unchanged
	Direction	P6DIR	036h	Read/write	Reset with PUC
	Port Select	P6SEL	037h	Read/write	Reset with PUC

# Watchdog Timer, Watchdog Timer+

---

---

---

---

The watchdog timer is a 16-bit timer that can be used as a watchdog or as an interval timer. This chapter describes the watchdog timer. The watchdog timer is implemented in all MSP430x4xx devices, except those with the enhanced watchdog timer, WDT+. The WDT+ is implemented in the MSP430x42x, MSP430FE42x, and MSP430F42x0 devices.

<b>Topic</b>	<b>Page</b>
<b>10.1 Watchdog Timer Introduction</b> .....	<b>10-2</b>
<b>10.2 Watchdog Timer Operation</b> .....	<b>10-4</b>
<b>10.2 Watchdog Timer Registers</b> .....	<b>10-7</b>

## 10.1 Watchdog Timer Introduction

The primary function of the watchdog timer (WDT) module is to perform a controlled system restart after a software problem occurs. If the selected time interval expires, a system reset is generated. If the watchdog function is not needed in an application, the module can be configured as an interval timer and can generate interrupts at selected time intervals.

Features of the watchdog timer module include:

- Four software-selectable time intervals
- Watchdog mode
- Interval mode
- Access to WDT control register is password protected
- Control of  $\overline{\text{RST}}$ /NMI pin function
- Selectable clock source
- Can be stopped to conserve power
- Clock fail-safe feature in WDT+

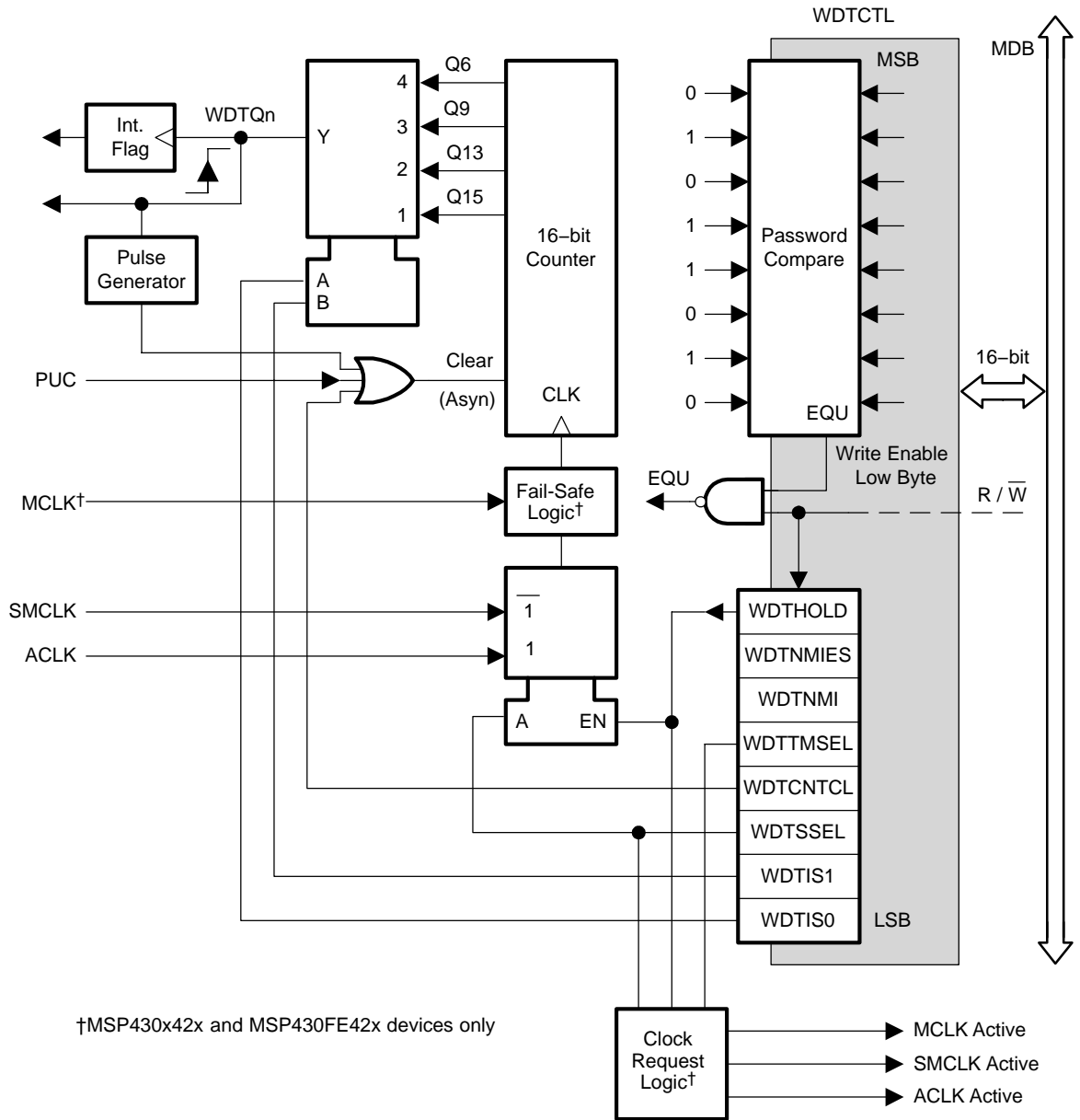
The WDT block diagram is shown in Figure 10–1.

**Note: Watchdog Timer Powers Up Active**

After a PUC, the WDT module is automatically configured in the watchdog mode with an initial ~32-ms reset interval using the DCOCLK. The user must setup or halt the WDT prior to the expiration of the initial reset interval.



Figure 10–1. Watchdog Timer Block Diagram



## 10.2 Watchdog Timer Operation

The WDT module can be configured as either a watchdog or interval timer with the WDTCTL register. The WDTCTL register also contains control bits to configure the  $\overline{\text{RST}}/\text{NMI}$  pin. WDTCTL is a 16-bit, password-protected, read/write register. Any read or write access must use word instructions and write accesses must include the write password 05Ah in the upper byte. Any write to WDTCTL with any value other than 05Ah in the upper byte is a security key violation and triggers a PUC system reset regardless of timer mode. Any read of WDTCTL reads 069h in the upper byte.

### 10.2.1 Watchdog Timer Counter

The watchdog timer counter (WDCNT) is a 16-bit up-counter that is not directly accessible by software. The WDCNT is controlled and time intervals selected through the watchdog timer control register WDTCTL.

The WDCNT can be sourced from ACLK or SMCLK. The clock source is selected with the WDTSSSEL bit.

### 10.2.2 Watchdog Mode

After a PUC condition, the WDT module is configured in the watchdog mode with an initial ~32-ms reset interval using the DCOCLK. The user must setup, halt, or clear the WDT prior to the expiration of the initial reset interval or another PUC will be generated. When the WDT is configured to operate in watchdog mode, either writing to WDTCTL with an incorrect password, or expiration of the selected time interval triggers a PUC. A PUC resets the WDT to its default condition and configures the  $\overline{\text{RST}}/\text{NMI}$  pin to reset mode.

### 10.2.3 Interval Timer Mode

Setting the WDTTMSSEL bit to 1 selects the interval timer mode. This mode can be used to provide periodic interrupts. In interval timer mode, the WDTIFG flag is set at the expiration of the selected time interval. A PUC is not generated in interval timer mode at expiration of the selected timer interval and the WDTIFG enable bit WDTIE remains unchanged.

When the WDTIE bit and the GIE bit are set, the WDTIFG flag requests an interrupt. The WDTIFG interrupt flag is automatically reset when its interrupt request is serviced, or may be reset by software. The interrupt vector address in interval timer mode is different from that in watchdog mode.

**Note: Modifying the Watchdog Timer**

The WDT interval should be changed together with WDCNTCL = 1 in a single instruction to avoid an unexpected immediate PUC or interrupt.

The WDT should be halted before changing the clock source to avoid a possible incorrect interval.

## 10.2.4 Watchdog Timer Interrupts

The WDT uses two bits in the SFRs for interrupt control.

- The WDT interrupt flag, WDTIFG, located in IFG1.0
- The WDT interrupt enable, WDTIE, located in IE1.0

When using the WDT in the watchdog mode, the WDTIFG flag sources a reset vector interrupt. The WDTIFG can be used by the reset interrupt service routine to determine if the watchdog caused the device to reset. If the flag is set, then the watchdog timer initiated the reset condition either by timing out or by a security key violation. If WDTIFG is cleared, the reset was caused by a different source.

When using the WDT in interval timer mode, the WDTIFG flag is set after the selected time interval and requests a WDT interval timer interrupt if the WDTIE and the GIE bits are set. The interval timer interrupt vector is different from the reset vector used in watchdog mode. In interval timer mode, the WDTIFG flag is reset automatically when the interrupt is serviced, or can be reset with software.

## 10.2.5 WDT+ Enhancements

The WDT+ module provides enhanced functionality over the WDT. The WDT+ provides a fail-safe clocking feature assuring the clock to the WDT+ cannot be disabled while in watchdog mode. This means the low-power modes may be affected by the choice for the WDT+ clock. For example, if ACLK is the WDT+ clock source, LPM4 will not be available, because the WDT+ will prevent ACLK from being disabled. Also, if ACLK or SMCLK fail while sourcing the WDT+, the WDT+ clock source is automatically switched to MCLK. In this case, if MCLK is sourced from a crystal, and the crystal has failed, the FLL+ fail-safe feature will activate the DCO and use it as the source for MCLK.

When the WDT+ module is used in interval timer mode, there is no fail-safe feature for the clock source.

## 10.2.6 Operation in Low-Power Modes

The MSP430 devices have several low-power modes. Different clock signals are available in different low-power modes. The requirements of the user's application and the type of clocking used determine how the WDT should be configured. For example, the WDT should not be configured in watchdog mode with SMCLK as its clock source if the user wants to use low-power mode 3 because SMCLK is not active in LPM3 and the WDT would not function. In this case with the WDT+ SMCLK would remain enabled increasing the current consumption of LPM3. When the watchdog timer is not required, the WDT\_HOLD bit can be used to hold the WDT\_CNT, reducing power consumption.

## 10.2.7 Software Examples

Any write operation to WDTCTL must be a word operation with 05Ah (WDTPW) in the upper byte:

```
; Periodically clear an active watchdog
    MOV #WDTPW+WDTCNTCL,&WDTCTL
;
; Change watchdog timer interval
    MOV #WDTPW+WDTCNTL+SSEL,&WDTCTL
;
; Stop the watchdog
    MOV #WDTPW+WDT_HOLD,&WDTCTL
;
; Change WDT to interval timer mode, clock/8192 interval
    MOV #WDTPW+WDTCNTCL+WDTTMSSEL+WDTIS0,&WDTCTL
```

### 10.3 Watchdog Timer Registers

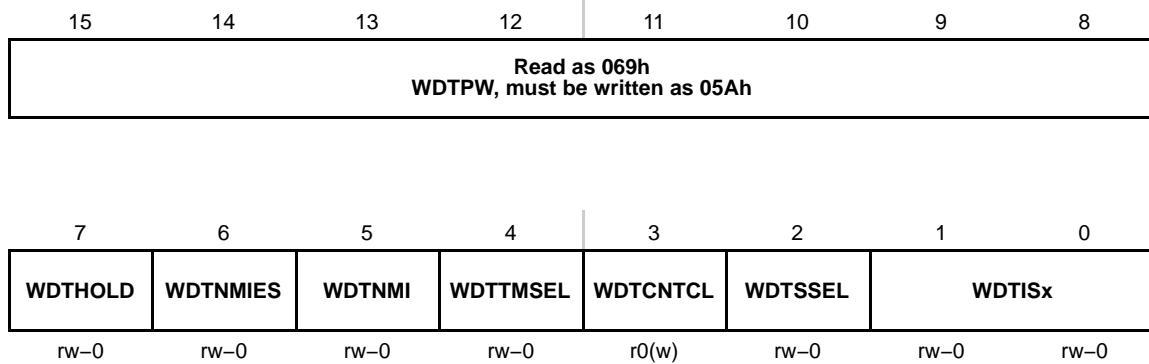
The watchdog timer module registers are listed in Table 10–1.

*Table 10–1. Watchdog Timer Registers*

<b>Register</b>	<b>Short Form</b>	<b>Register Type</b>	<b>Address</b>	<b>Initial State</b>
Watchdog timer control register	WDTCTL	Read/write	0120h	06900h with PUC
SFR interrupt enable register 1	IE1	Read/write	0000h	Reset with PUC
SFR interrupt flag register 1	IFG1	Read/write	0002h	Reset with PUC†

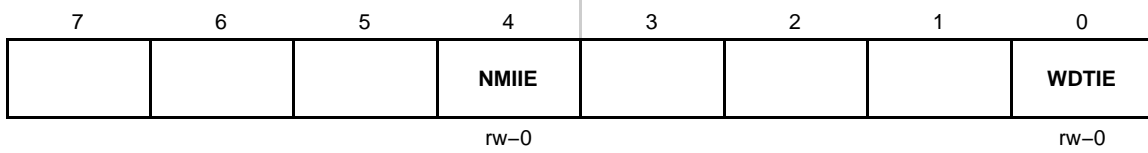
† WDTIFG is reset with POR

### WDTCTL, Watchdog Timer Register



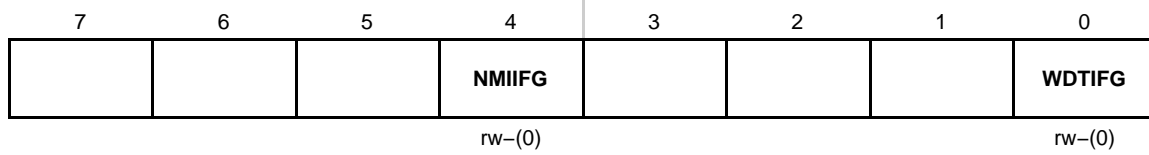
- WDTPW**    Bits 15-8    Watchdog timer password. Always read as 069h. Must be written as 05Ah, or a PUC will be generated.
- WDTHOLD**    Bit 7    Watchdog timer hold. This bit stops the watchdog timer. Setting WDTHOLD = 1 when the WDT is not in use conserves power.  
 0    Watchdog timer is not stopped  
 1    Watchdog timer is stopped
- WDTNMIES**    Bit 6    Watchdog timer NMI edge select. This bit selects the interrupt edge for the NMI interrupt when WDTNMI = 1. Modifying this bit can trigger an NMI. Modify this bit when WDTNMI = 0 to avoid triggering an accidental NMI.  
 0    NMI on rising edge  
 1    NMI on falling edge
- WDTNMI**    Bit 5    Watchdog timer NMI select. This bit selects the function for the  $\overline{\text{RST}}$ /NMI pin.  
 0    Reset function  
 1    NMI function
- WDTTMSSEL**    Bit 4    Watchdog timer mode select  
 0    Watchdog mode  
 1    Interval timer mode
- WDTCNTCL**    Bit 3    Watchdog timer counter clear. Setting WDTCNTCL = 1 clears the count value to 0000h. WDTCNTCL is automatically reset.  
 0    No action  
 1    WDTCNT = 0000h
- WDTSSSEL**    Bit 2    Watchdog timer clock source select  
 0    SMCLK  
 1    ACLK
- WDTISx**    Bits 1-0    Watchdog timer interval select. These bits select the watchdog timer interval to set the WDTIFG flag and/or generate a PUC.  
 00    Watchdog clock source /32768  
 01    Watchdog clock source /8192  
 10    Watchdog clock source /512  
 11    Watchdog clock source /64

**IE1, Interrupt Enable Register 1**



	Bits	7-5	These bits may be used by other modules. See device-specific datasheet.
<b>NMIIE</b>	Bit	4	<p>NMI interrupt enable. This bit enables the NMI interrupt. Because other bits in IE1 may be used for other modules, it is recommended to set or clear this bit using <code>BIS.B</code> or <code>BIC.B</code> instructions, rather than <code>MOV.B</code> or <code>CLR.B</code> instructions.</p> <p>0    Interrupt not enabled 1    Interrupt enabled</p>
	Bits	3-1	These bits may be used by other modules. See device-specific datasheet.
<b>WDTIE</b>	Bit	0	<p>Watchdog timer interrupt enable. This bit enables the WDTIFG interrupt for interval timer mode. It is not necessary to set this bit for watchdog mode. Because other bits in IE1 may be used for other modules, it is recommended to set or clear this bit using <code>BIS.B</code> or <code>BIC.B</code> instructions, rather than <code>MOV.B</code> or <code>CLR.B</code> instructions.</p> <p>0    Interrupt not enabled 1    Interrupt enabled</p>

### IFG1, Interrupt Flag Register 1



Bits 7-5: These bits may be used by other modules. See device-specific datasheet.

**NMIIFG** Bit 4: NMI interrupt flag. NMIIFG must be reset by software. Because other bits in IFG1 may be used for other modules, it is recommended to clear NMIIFG by using `BIS.B` or `BIC.B` instructions, rather than `MOV.B` or `CLR.B` instructions.

0 No interrupt pending  
1 Interrupt pending

Bits 3-1: These bits may be used by other modules. See device-specific datasheet.

**WDTIFG** Bit 0: Watchdog timer interrupt flag. In watchdog mode, WDTIFG remains set until reset by software. In interval mode, WDTIFG is reset automatically by servicing the interrupt, or can be reset by software. Because other bits in IFG1 may be used for other modules, it is recommended to clear WDTIFG by using `BIS.B` or `BIC.B` instructions, rather than `MOV.B` or `CLR.B` instructions.

0 No interrupt pending  
1 Interrupt pending



# Basic Timer1

---

---

---

---

The Basic Timer1 module is two independent, cascadable 8-bit timers. This chapter describes the Basic Timer1. Basic Timer1 is implemented in all MSP430x4xx devices.

<b>Topic</b>	<b>Page</b>
<b>11.1 Basic Timer1 Introduction</b> .....	<b>11-2</b>
<b>11.2 Basic Timer1 Operation</b> .....	<b>11-4</b>
<b>11.3 Basic Timer1 Registers</b> .....	<b>11-6</b>

## 11.1 Basic Timer1 Introduction

The Basic Timer1 supplies LCD timing and low frequency time intervals. The Basic Timer1 is two independent 8-bit timers that can also be cascaded to form one 16-bit timer function.

Some uses for the Basic Timer1 include:

- Real-time clock (RTC) function
- Software time increments

Basic Timer1 features include:

- Selectable clock source
- Two independent, cascadable 8-bit timers
- Interrupt capability
- LCD control signal generation

The Basic Timer1 block diagram is shown in Figure 11–1.

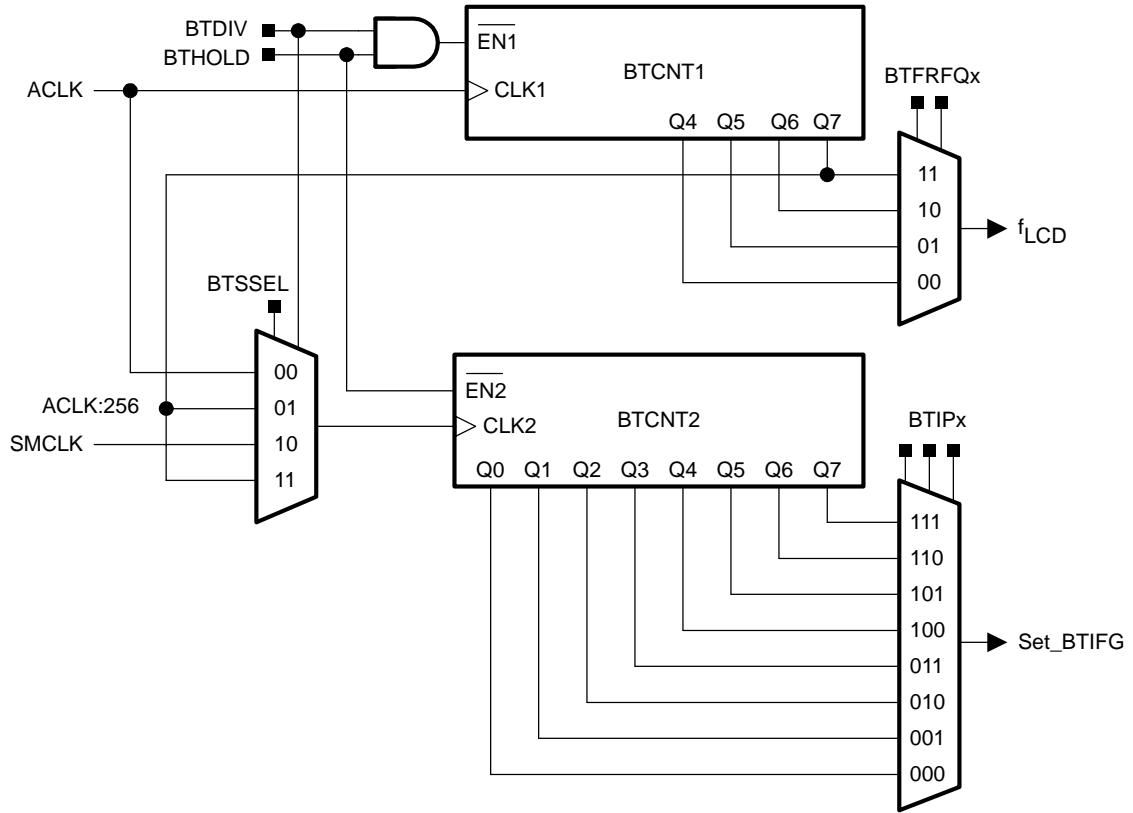
---

**Note: Basic Timer1 Initialization**

The Basic Timer1 module registers have no initial condition. These registers must be configured by user software before use.

---

Figure 11–1. Basic Timer1 Block Diagram



## 11.2 Basic Timer1 Operation

The Basic Timer1 module can be configured as two 8-bit timers or one 16-bit timer with the BTCTL register. The BTCTL register is an 8-bit, read/write register. Any read or write access must use byte instructions. The Basic Timer1 controls the LCD frame frequency with BTCNT1.

### 11.2.1 Basic Timer1 Counter One

The Basic Timer1 counter one, BTCNT1, is an 8-bit timer/counter directly accessible by software. BTCNT1 is incremented with ACLK and provides the frame frequency for the LCD controller. BTCNT1 can be stopped by setting the BTHOLD and BTDIV bits.

### 11.2.2 Basic Timer1 Counter Two

The Basic Timer1 counter two, BTCNT2, is an 8-bit timer/counter directly accessible by software. BTCNT2 can be sourced from ACLK or SMCLK, or ACLK/256 when cascaded with BTCNT1. The BTCNT2 clock source is selected with the BTSSEL and BTDIV bits. BTCNT2 can be stopped to reduce power consumption by setting the HOLD bit.

BTCNT2 sources the Basic Timer1 interrupt, BTIFG. The interrupt interval is selected with the BTIPx bits

**Note: Reading or Writing BTCNT1 and BTCNT2**

When the CPU clock and counter clock are asynchronous any read from BTCNT1 or BTCNT2 may be unpredictable. Any write to BTCNT1 or BTCNT2 take effect immediately.

### 11.2.3 16-bit Counter Mode

The 16-bit timer/counter mode is selected when control the BTDIV bit is set. In this mode, BTCNT1 is cascaded with BTCNT2. The clock source of BTCNT1 is ACLK, and the clock source of BTCNT2 is ACLK/256.

### 11.2.4 Basic Timer1 Operation: Signal $f_{LCD}$

The LCD controller (but not the LCDA controller) uses the  $f_{LCD}$  signal from the BTCNT1 to generate the timing for common and segment lines. ACLK sources BTCNT1 and is assumed to be 32768 Hz for generating  $f_{LCD}$ . The  $f_{LCD}$  frequency is selected with the BTFRFQx bits and can be ACLK/256, ACLK/128, ACLK/64, or ACLK/32. The proper  $f_{LCD}$  frequency depends on the LCD's frame frequency and the LCD multiplex rate and is calculated by:

$$f_{LCD} = 2 \times \text{mux} \times f_{\text{Frame}}$$

For example, to calculate  $f_{LCD}$  for a 3-mux LCD, with a frame frequency of 30 - 100Hz:

$$f_{\text{Frame}} \text{ (from LCD datasheet)} = 30 - 100 \text{ Hz}$$

$$f_{LCD} = 2 \times 3 \times f_{\text{Frame}}$$

$$f_{LCD(\text{min})} = 180 \text{ Hz}$$

$$f_{LCD(\text{max})} = 600 \text{ Hz}$$

$$\text{select } f_{LCD} = 32768/128 = 256 \text{ Hz or } 32768/64 = 512 \text{ Hz}$$

The LCD\_A controller does not use the Basic Timer1 for  $f_{LCD}$  generation. See the *LCD Controller* and *LCD\_A Controller* chapters for more details on the LCD controllers.

### 11.2.5 Basic Timer1 Interrupts

The Basic Timer1 uses two bits in the SFRs for interrupt control.

- Basic Timer1 interrupt flag, BTIFG, located in IFG2.7
- Basic Timer1 interrupt enable, BTIE, located in IE2.7

The BTIFG flag is set after the selected time interval and requests a Basic Timer1 interrupt if the BTIE and the GIE bits are set. The BTIFG flag is reset automatically when the interrupt is serviced, or can be reset with software.

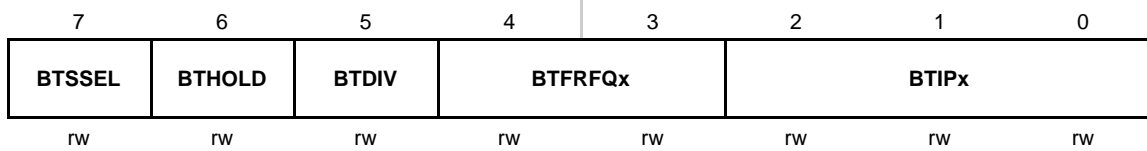
### 11.3 Basic Timer1 Registers

The watchdog timer module registers are listed in Table 11–1.

Table 11–1. Basic Timer1 Registers

Register	Short Form	Register Type	Address	Initial State
Basic Timer1 Control	BTCTL	Read/write	040h	Unchanged
Basic Timer1 Counter 1	BTCNT1	Read/write	046h	Unchanged
Basic Timer1 Counter 2	BTCNT2	Read/write	047h	Unchanged
SFR interrupt flag register 2	IFG2	Read/write	001h	Reset with PUC
SFR interrupt enable register 2	IE2	Read/write	003h	Reset with PUC

**Note:** The Basic Timer1 registers should be configured at power-up. There is no initial state for BTCTL, BTCNT1, or BTCNT2

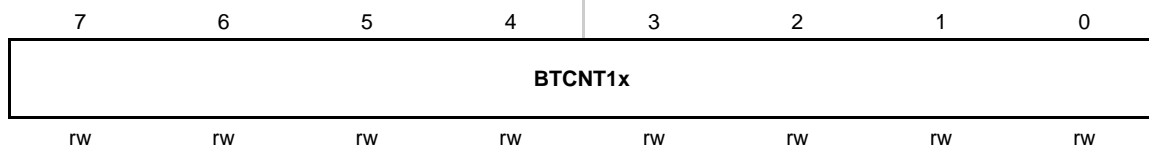
**BTCTL, Basic Timer1 Control Register**

<b>BTSSSEL</b>	Bit 7	BTCNT2 clock select. This bit, together with the BTDIV bit, selects the clock source for BTCNT2. See the description for BTDIV.
<b>BTHOLD</b>	Bit 6	Basic Timer1 Hold. 0 BTCNT1 and BTCNT2 are operational 1 BTCNT1 is held if BTDIV=1 BTCNT2 is held
<b>BTDIV</b>	Bit 5	Basic Timer1 clock divide. This bit together with the BTSSSEL bit, selects the clock source for BTCNT2.

BTSSSEL	BTDIV	BTCNT2 Clock Source
0	0	ACLK
0	1	ACLK/256
1	0	SMCLK
1	1	ACLK/256

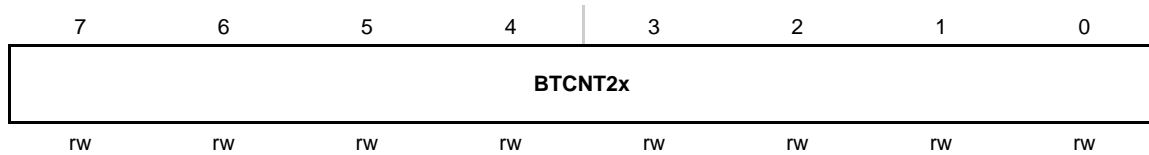
<b>BTFRFQx</b>	Bits 4–3	$f_{LCD}$ frequency. These bits control the LCD update frequency. 00 $f_{ACLK}/32$ 01 $f_{ACLK}/64$ 10 $f_{ACLK}/128$ 11 $f_{ACLK}/256$
<b>BTIPx</b>	Bits 2–0	Basic Timer1 Interrupt Interval. 000 $f_{CLK2}/2$ 001 $f_{CLK2}/4$ 010 $f_{CLK2}/8$ 011 $f_{CLK2}/16$ 100 $f_{CLK2}/32$ 101 $f_{CLK2}/64$ 110 $f_{CLK2}/128$ 111 $f_{CLK2}/256$

### BTCNT1, Basic Timer1 Counter 1



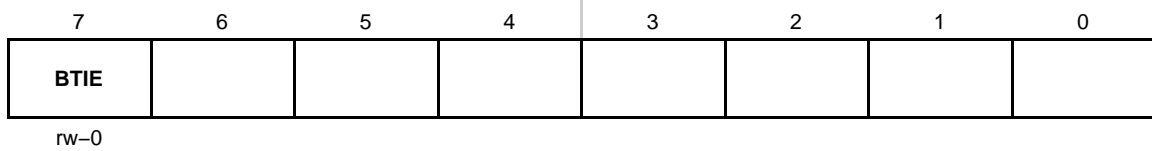
**BTCNT1x** Bits 7-0 BTCNT1 register. The BTCNT1 register is the count of BTCNT1.

### BTCNT2, Basic Timer1 Counter 2

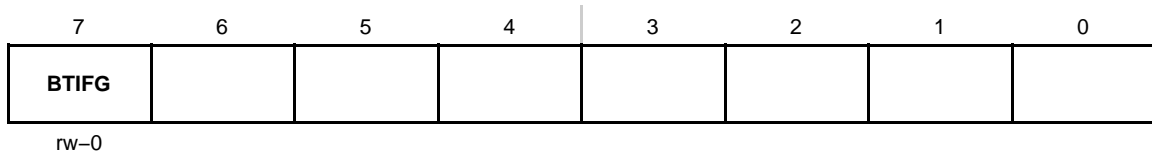


**BTCNT2x** Bits 7-0 BTCNT2 register. The BTCNT2 register is the count of BTCNT2.



**IE2, Interrupt Enable Register 2**

<b>BTIE</b>	Bit 7	Basic Timer1 interrupt enable. This bit enables the BTIFG interrupt. Because other bits in IE2 may be used for other modules, it is recommended to set or clear this bit using <code>BIS.B</code> or <code>BIC.B</code> instructions, rather than <code>MOV.B</code> or <code>CLR.B</code> instructions. 0    Interrupt not enabled 1    Interrupt enabled
	Bits 6-1	These bits may be used by other modules. See device-specific datasheet.

**IFG2, Interrupt Flag Register 2**

<b>BTIFG</b>	Bit 7	Basic Timer1 interrupt flag. Because other bits in IFG2 may be used for other modules, it is recommended to clear BTIFG automatically by servicing the interrupt, or by using <code>BIS.B</code> or <code>BIC.B</code> instructions, rather than <code>MOV.B</code> or <code>CLR.B</code> instructions. 0    No interrupt pending 1    Interrupt pending
	Bits 6-1	These bits may be used by other modules. See device-specific datasheet.

# Timer\_A

---

---

---

---

Timer\_A is a 16-bit timer/counter with multiple capture/compare registers. This chapter describes Timer\_A. Timer\_A3 (three capture/compare registers) is implemented in all MSP430x4xx devices. Timer1\_A5 (five capture/compare registers) is also implemented on MSP430x415, MSP430x417, and MSP430xW42x devices.

<b>Topic</b>	<b>Page</b>
<b>12.1 Timer_A Introduction</b> .....	<b>12-2</b>
<b>12.2 Timer_A Operation</b> .....	<b>12-4</b>
<b>12.3 Timer_A Registers</b> .....	<b>12-19</b>

## 12.1 Timer\_A Introduction

Timer\_A is a 16-bit timer/counter with three or five capture/compare registers. Timer\_A can support multiple capture/compares, PWM outputs, and interval timing. Timer\_A also has extensive interrupt capabilities. Interrupts may be generated from the counter on overflow conditions and from each of the capture/compare registers.

Timer\_A features include:

- Asynchronous 16-bit timer/counter with four operating modes
- Selectable and configurable clock source
- Three or five configurable capture/compare registers
- Configurable outputs with PWM capability
- Asynchronous input and output latching
- Interrupt vector register for fast decoding of all Timer\_A interrupts

The block diagram of Timer\_A is shown in Figure 12–1.

---

**Note: Use of the Word *Count***

*Count* is used throughout this chapter. It means the counter must be in the process of counting for the action to take place. If a particular value is directly written to the counter, then an associated action will not take place.

---

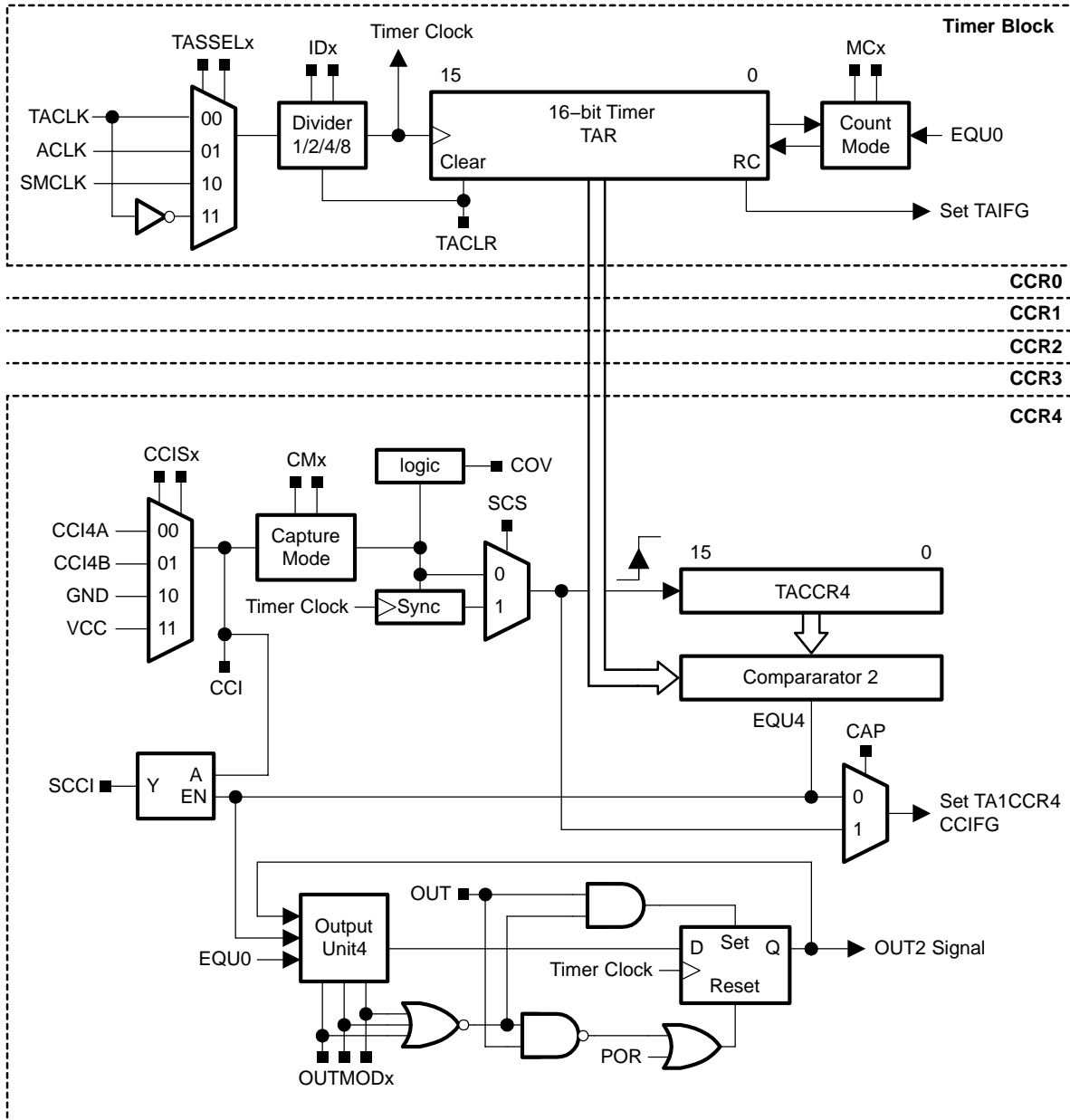
---

**Note: Second Timer\_A On Select Devices**

MSP430x415, MSP430x417, and MSP430xW42x devices implement a second Timer\_A with five capture/compare registers. On these devices, both Timer\_A modules are identical in function, except for the additional capture/compare registers.

---

Figure 12-1. Timer\_A Block Diagram



## 12.2 Timer\_A Operation

The Timer\_A module is configured with user software. The setup and operation of Timer\_A is discussed in the following sections.

### 12.2.1 16-Bit Timer Counter

The 16-bit timer/counter register, TAR, increments or decrements (depending on mode of operation) with each rising edge of the clock signal. TAR can be read or written with software. Additionally, the timer can generate an interrupt when it overflows.

TAR may be cleared by setting the TACLRL bit. Setting TACLRL also clears the clock divider and count direction for up/down mode.

---

**Note: Modifying Timer\_A Registers**

It is recommended to stop the timer before modifying its operation (with exception of the interrupt enable, interrupt flag, and TACLRL) to avoid errant operating conditions.

When the TACLK is asynchronous to the CPU clock, any read from TAR should occur while the timer is not operating or the results may be unpredictable. Alternatively, the timer may be read multiple times while operating, and a majority vote taken in software to determine the correct reading. Any write to TAR will take effect immediately.

---

### Clock Source Select and Divider

The timer clock TACLK can be sourced from ACLK, SMCLK, or externally via TACLK or INCLK. The clock source is selected with the TASSELx bits. The selected clock source may be passed directly to the timer or divided by 2, 4, or 8, using the IDx bits. The TACLK divider is reset when TACLRL is set.

## 12.2.2 Starting the Timer

The timer may be started, or restarted in the following ways:

- The timer counts when MCx > 0 and the clock source is active.
- When the timer mode is either up or up/down, the timer may be stopped by writing 0 to TACCR0. The timer may then be restarted by writing a nonzero value to TACCR0. In this scenario, the timer starts incrementing in the up direction from zero.

## 12.2.3 Timer Mode Control

The timer has four modes of operation as described in Table 12–1: stop, up, continuous, and up/down. The operating mode is selected with the MCx bits.

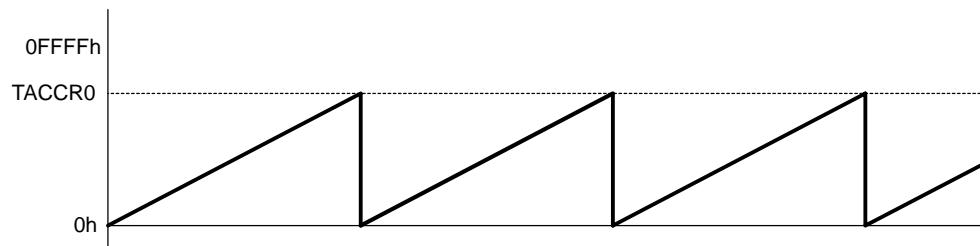
Table 12–1. Timer Modes

MCx	Mode	Description
00	Stop	The timer is halted.
01	Up	The timer repeatedly counts from zero to the value of TACCR0
10	Continuous	The timer repeatedly counts from zero to 0FFFFh.
11	Up/down	The timer repeatedly counts from zero up to the value of TACCR0 and back down to zero.

## Up Mode

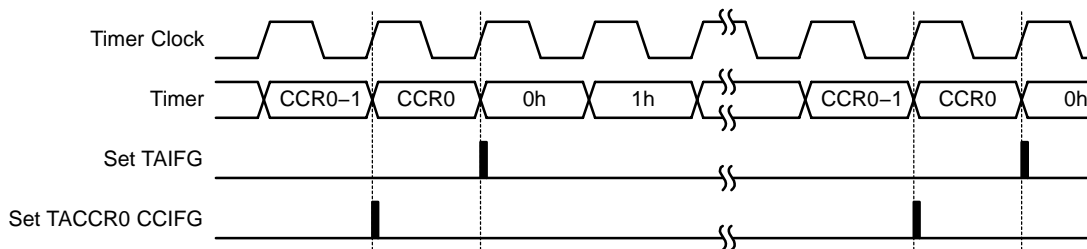
The up mode is used if the timer period must be different from 0FFFFh counts. The timer repeatedly counts up to the value of compare register TACCR0, which defines the period, as shown in Figure 12–2. The number of timer counts in the period is TACCR0+1. When the timer value equals TACCR0 the timer restarts counting from zero. If up mode is selected when the timer value is greater than TACCR0, the timer immediately restarts counting from zero.

Figure 12–2. Up Mode



The TACCR0 CCIFG interrupt flag is set when the timer *counts* to the TACCR0 value. The TAIFG interrupt flag is set when the timer *counts* from TACCR0 to zero. Figure 12–3 shows the flag set cycle.

Figure 12–3. Up Mode Flag Setting



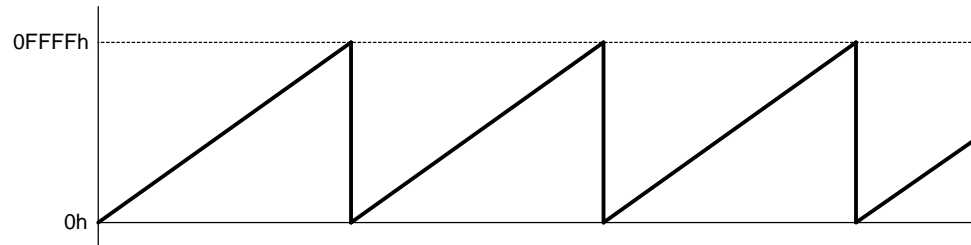
## Changing the Period Register TACCR0

When changing TACCR0 while the timer is running, if the new period is greater than or equal to the old period, or greater than the current count value, the timer counts up to the new period. If the new period is less than the current count value, the timer rolls to zero. However, one additional count may occur before the counter rolls to zero.

## Continuous Mode

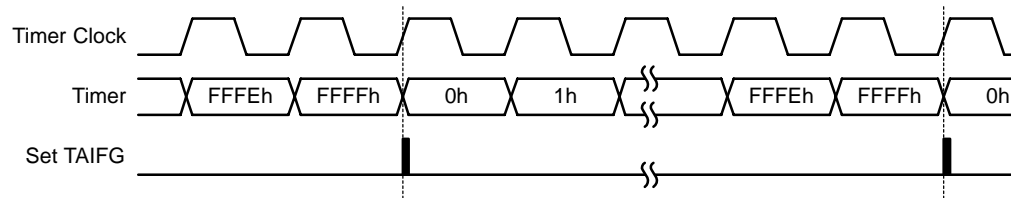
In the continuous mode, the timer repeatedly counts up to 0FFFFh and restarts from zero as shown in Figure 12–4. The capture/compare register TACCR0 works the same way as the other capture/compare registers.

Figure 12–4. Continuous Mode



The TAIFG interrupt flag is set when the timer *counts* from 0FFFFh to zero. Figure 12–5 shows the flag set cycle.

Figure 12–5. Continuous Mode Flag Setting

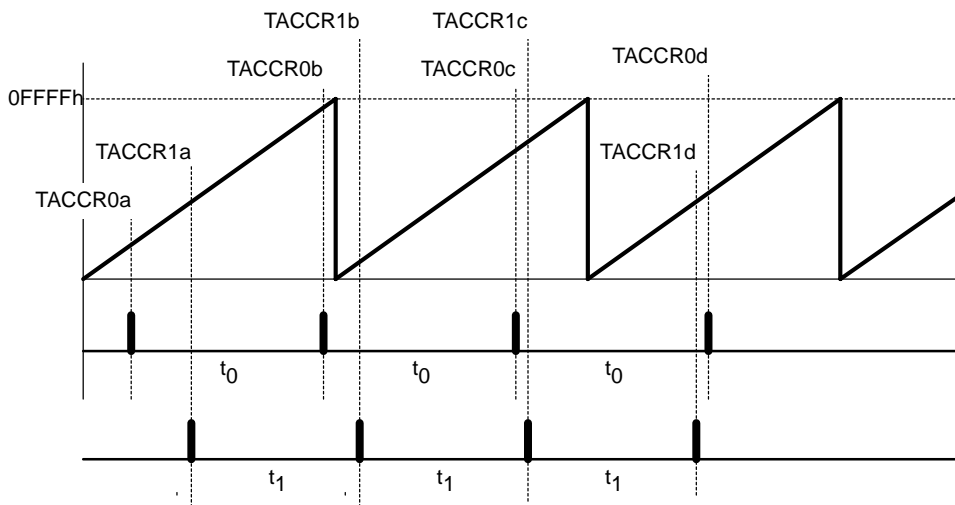




### Use of the Continuous Mode

The continuous mode can be used to generate independent time intervals and output frequencies. Each time an interval is completed, an interrupt is generated. The next time interval is added to the TACCRx register in the interrupt service routine. Figure 12–6 shows two separate time intervals  $t_0$  and  $t_1$  being added to the capture/compare registers. In this usage, the time interval is controlled by hardware, not software, without impact from interrupt latency. Up to three (Timer\_A3) or five (Timer\_A5) independent time intervals or output frequencies can be generated using capture/compare registers.

Figure 12–6. Continuous Mode Time Intervals

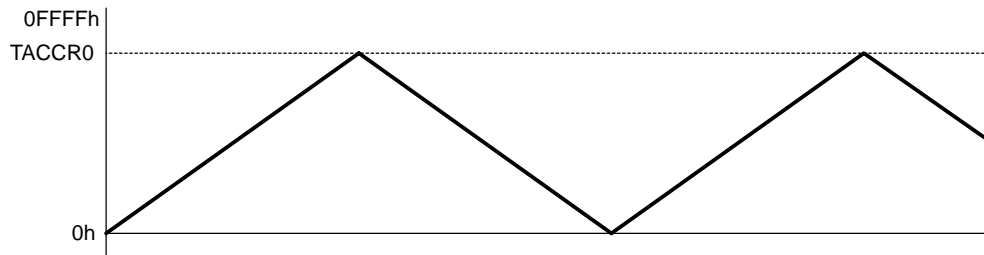


Time intervals can be produced with other modes as well, where TACCR0 is used as the period register. Their handling is more complex since the sum of the old TACCRx data and the new period can be higher than the TACCR0 value. When the previous TACCRx value plus  $t_x$  is greater than the TACCR0 data, the TACCR0 value must be subtracted to obtain the correct time interval.

## Up/Down Mode

The up/down mode is used if the timer period must be different from 0FFFFh counts, and if symmetrical pulse generation is needed. The timer repeatedly counts up to the value of compare register TACCR0 and back down to zero, as shown in Figure 12–7. The period is twice the value in TACCR0.

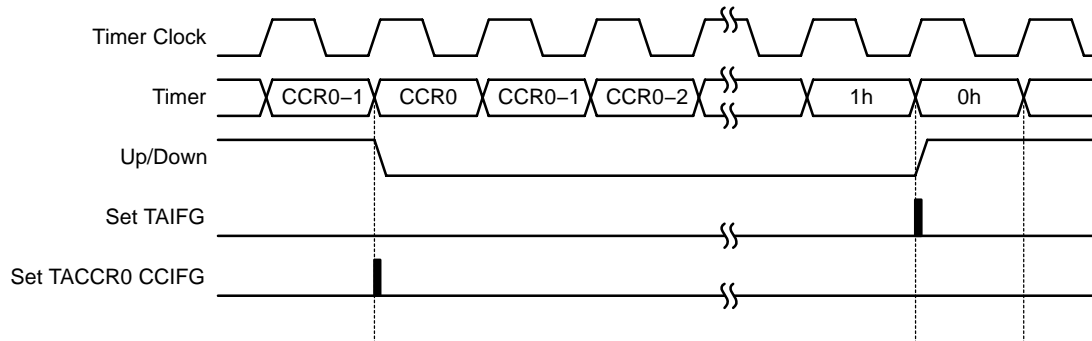
Figure 12–7. Up/Down Mode



The count direction is latched. This allows the timer to be stopped and then restarted in the same direction it was counting before it was stopped. If this is not desired, the TACLR bit must be set to clear the direction. The TACLR bit also clears the TAR value and the TACLK divider.

In up/down mode, the TACCR0 CCIFG interrupt flag and the TAIFG interrupt flag are set only once during a period, separated by 1/2 the timer period. The TACCR0 CCIFG interrupt flag is set when the timer *counts* from TACCR0–1 to TACCR0, and TAIFG is set when the timer completes *counting* down from 0001h to 0000h. Figure 12–8 shows the flag set cycle.

Figure 12–8. Up/Down Mode Flag Setting



### Changing the Period Register TACCR0

When changing TACCR0 while the timer is running, and counting in the down direction, the timer continues its descent until it reaches zero. The new period takes affect after the counter counts down to zero.

When the timer is counting in the up direction, and the new period is greater than or equal to the old period, or greater than the current count value, the timer counts up to the new period before counting down. When the timer is counting in the up direction, and the new period is less than the current count value, the timer begins counting down. However, one additional count may occur before the counter begins counting down.

### Use of the Up/Down Mode

The up/down mode supports applications that require dead times between output signals (See section *Timer\_A Output Unit*). For example, to avoid overload conditions, two outputs driving an H-bridge must never be in a high state simultaneously. In the example shown in Figure 12–9 the  $t_{dead}$  is:

$$t_{dead} = t_{timer} \times (TACCR1 - TACCR2)$$

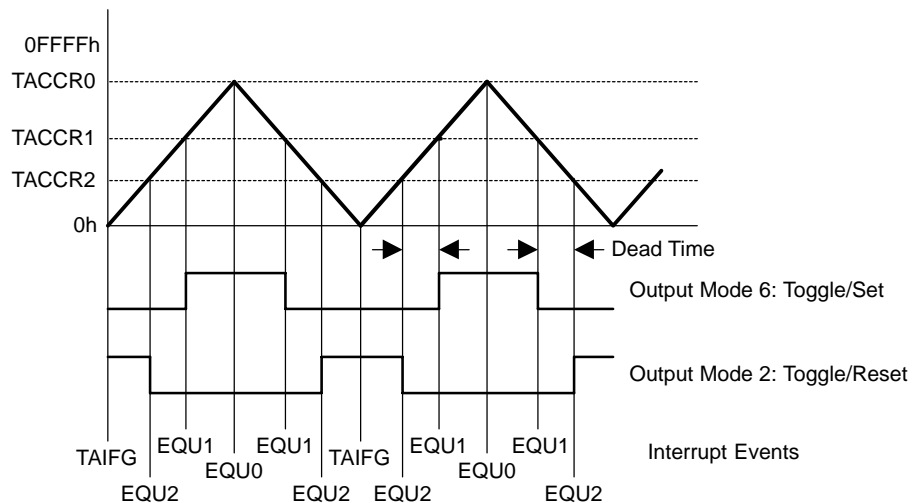
With:  $t_{dead}$  Time during which both outputs need to be inactive

$t_{timer}$  Cycle time of the timer clock

TACCRx Content of capture/compare register x

The TACCRx registers are not buffered. They update immediately when written to. Therefore, any required dead time will not be maintained automatically.

Figure 12–9. Output Unit in Up/Down Mode



## 12.2.4 Capture/Compare Blocks

Three or five identical capture/compare blocks, TACCRx, are present in Timer\_A. Any of the blocks may be used to capture the timer data, or to generate time intervals.

### Capture Mode

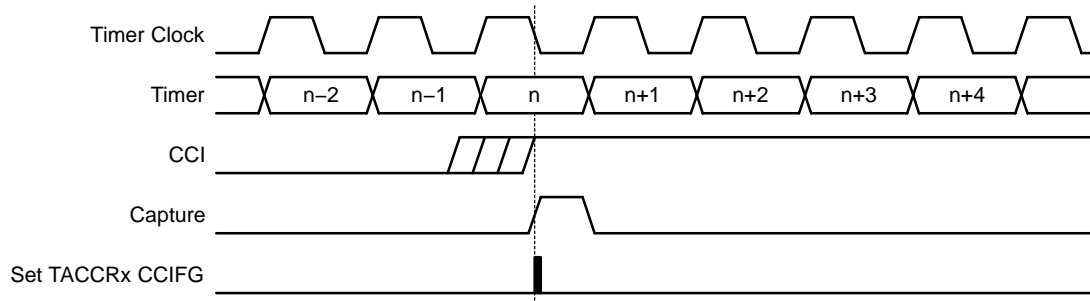
The capture mode is selected when CAP = 1. Capture mode is used to record time events. It can be used for speed computations or time measurements. The capture inputs CCIxA and CCIxB are connected to external pins or internal signals and are selected with the CCISx bits. The CMx bits select the capture edge of the input signal as rising, falling, or both. A capture occurs on the selected edge of the input signal. If a capture occurs:

- The timer value is copied into the TACCRx register
- The interrupt flag CCIFG is set

The input signal level can be read at any time via the CCI bit. MSP430x4xx family devices may have different signals connected to CCIxA and CCIxB. Refer to the device-specific datasheet for the connections of these signals.

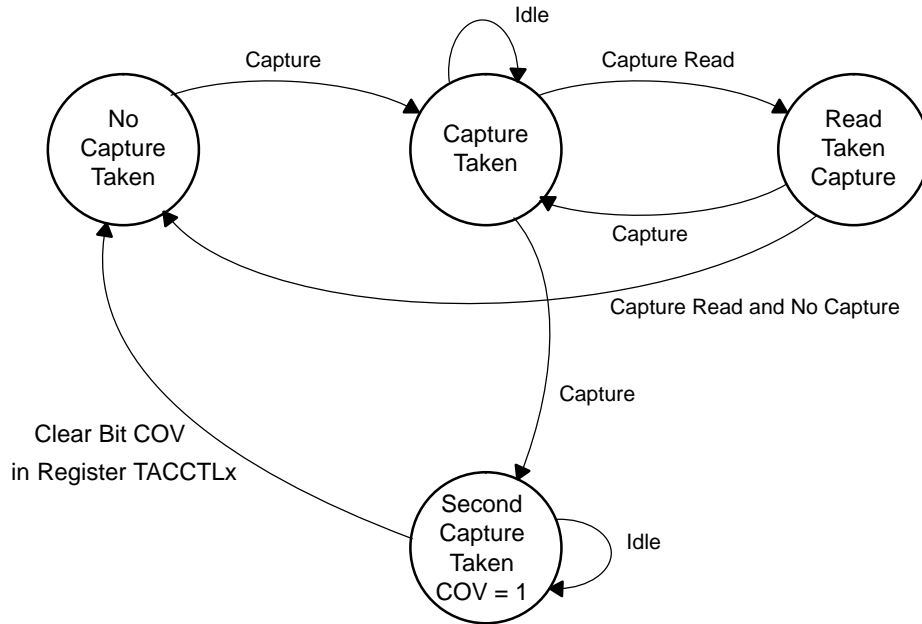
The capture signal can be asynchronous to the timer clock and cause a race condition. Setting the SCS bit will synchronize the capture with the next timer clock. Setting the SCS bit to synchronize the capture signal with the timer clock is recommended. This is illustrated in Figure 12–10.

Figure 12–10. Capture Signal (SCS=1)



Overflow logic is provided in each capture/compare register to indicate if a second capture was performed before the value from the first capture was read. Bit COV is set when this occurs as shown in Figure 12–11. COV must be reset with software.

Figure 12–11. Capture Cycle



**Capture Initiated by Software**

Captures can be initiated by software. The CMx bits can be set for capture on both edges. Software then sets CCIS1 = 1 and toggles bit CCIS0 to switch the capture signal between V<sub>CC</sub> and GND, initiating a capture each time CCIS0 changes state:

```

MOV    #CAP+SCS+CCIS1+CM_3,&TACCTLx ; Setup TACCTLx
XOR    #CCIS0,&TACCTLx                ; TACCTLx = TAR
  
```

**Compare Mode**

The compare mode is selected when CAP = 0. The compare mode is used to generate PWM output signals or interrupts at specific time intervals. When TAR counts to the value in a TACCRx:

- Interrupt flag CCIFG is set
- Internal signal EQUx = 1
- EQUx affects the output according to the output mode
- The input signal CCI is latched into SCCI

## 12.2.5 Output Unit

Each capture/compare block contains an output unit. The output unit is used to generate output signals such as PWM signals. Each output unit has eight operating modes that generate signals based on the EQU0 and EQUx signals.

### Output Modes

The output modes are defined by the OUTMODx bits and are described in Table 12–2. The OUTx signal is changed with the rising edge of the timer clock for all modes except mode 0. Output modes 2, 3, 6, and 7 are not useful for output unit 0 because EQUx = EQU0.

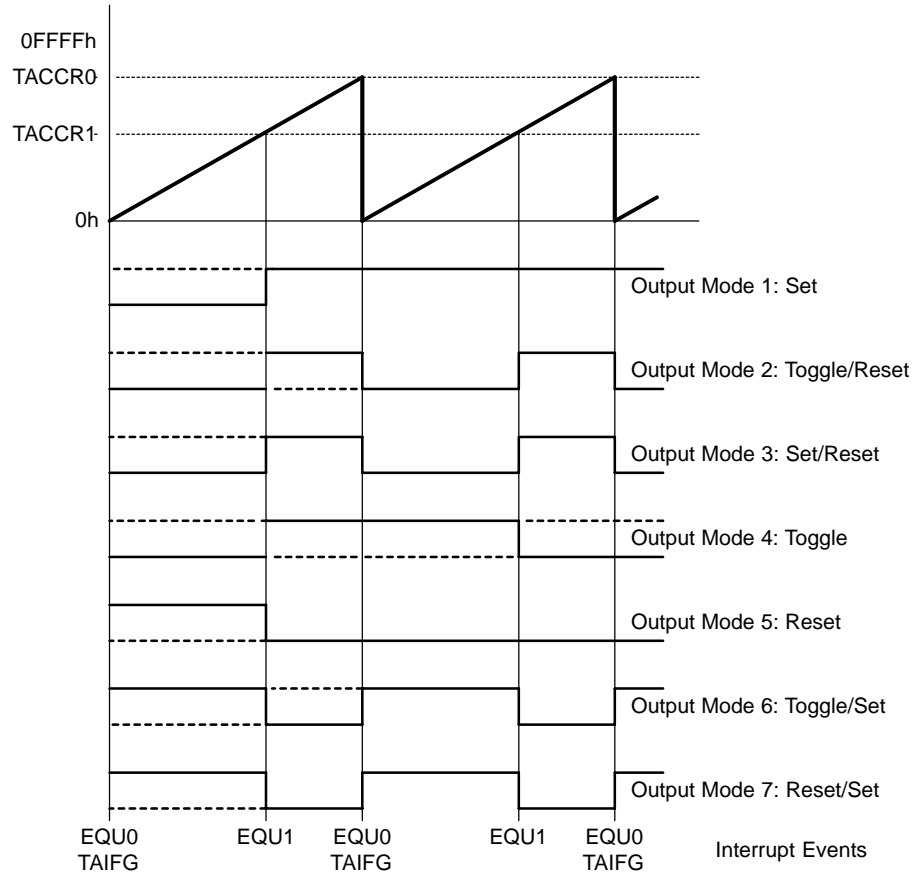
Table 12–2. Output Modes

OUTMODx	Mode	Description
000	Output	The output signal OUTx is defined by the OUTx bit. The OUTx signal updates immediately when OUTx is updated.
001	Set	The output is set when the timer <i>counts</i> to the TACCRx value. It remains set until a reset of the timer, or until another output mode is selected and affects the output.
010	Toggle/Reset	The output is toggled when the timer <i>counts</i> to the TACCRx value. It is reset when the timer <i>counts</i> to the TACCRO value.
011	Set/Reset	The output is set when the timer <i>counts</i> to the TACCRx value. It is reset when the timer <i>counts</i> to the TACCRO value.
100	Toggle	The output is toggled when the timer <i>counts</i> to the TACCRx value. The output period is double the timer period.
101	Reset	The output is reset when the timer <i>counts</i> to the TACCRx value. It remains reset until another output mode is selected and affects the output.
110	Toggle/Set	The output is toggled when the timer <i>counts</i> to the TACCRx value. It is set when the timer <i>counts</i> to the TACCRO value.
111	Reset/Set	The output is reset when the timer <i>counts</i> to the TACCRx value. It is set when the timer <i>counts</i> to the TACCRO value.

**Output Example—Timer in Up Mode**

The OUTx signal is changed when the timer *counts* up to the TACCRx value, and rolls from TACCR0 to zero, depending on the output mode. An example is shown in Figure 12–12 using TACCR0 and TACCR1.

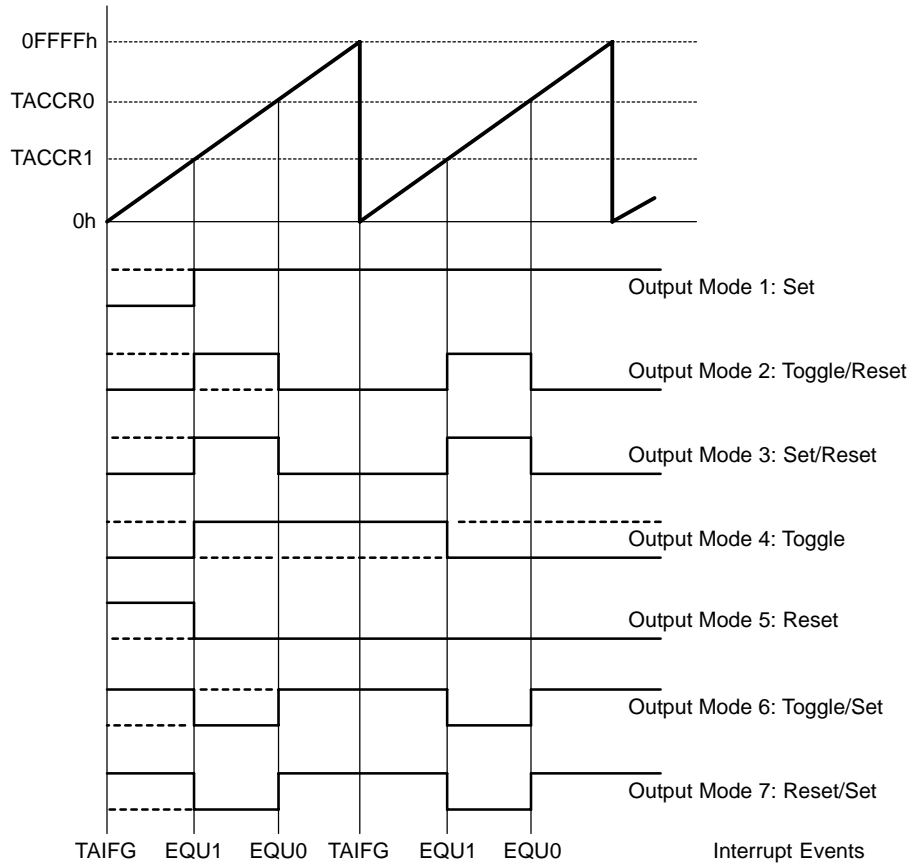
Figure 12–12. Output Example—Timer in Up Mode



**Output Example—Timer in Continuous Mode**

The OUTx signal is changed when the timer reaches the TACCRx and TACCR0 values, depending on the output mode. An example is shown in Figure 12–13 using TACCR0 and TACCR1.

Figure 12–13. Output Example—Timer in Continuous Mode

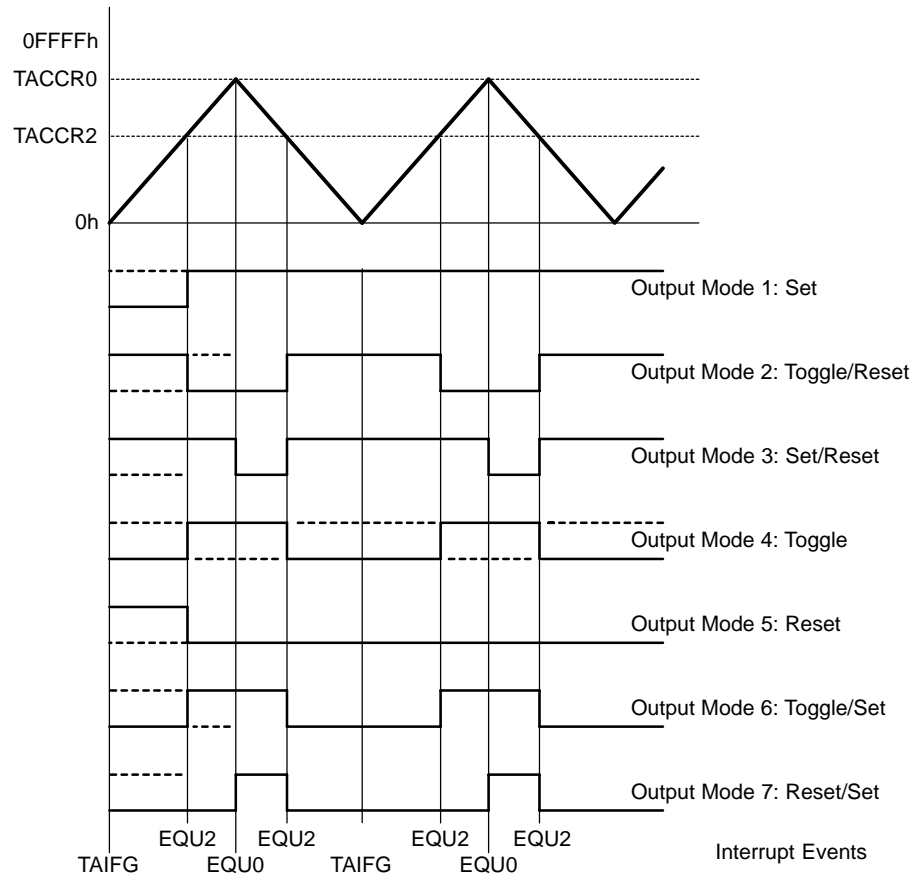




**Output Example—Timer in Up/Down Mode**

The OUTx signal changes when the timer equals TACCRx in either count direction and when the timer equals TACCR0, depending on the output mode. An example is shown in Figure 12–14 using TACCR0 and TACCR2.

Figure 12–14. Output Example—Timer in Up/Down Mode



**Note: Switching Between Output Modes**

When switching between output modes, one of the OUTMODx bits should remain set during the transition, unless switching to mode 0. Otherwise, output glitching can occur because a NOR gate decodes output mode 0. A safe method for switching between output modes is to use output mode 7 as a transition state:

```
BIS #OUTMOD_7,&TACCTLx ; Set output mode=7
BIC #OUTMODx,&TACCTLx ; Clear unwanted bits
```

## 12.2.6 Timer\_A Interrupts

Two interrupt vectors are associated with the 16-bit Timer\_A module:

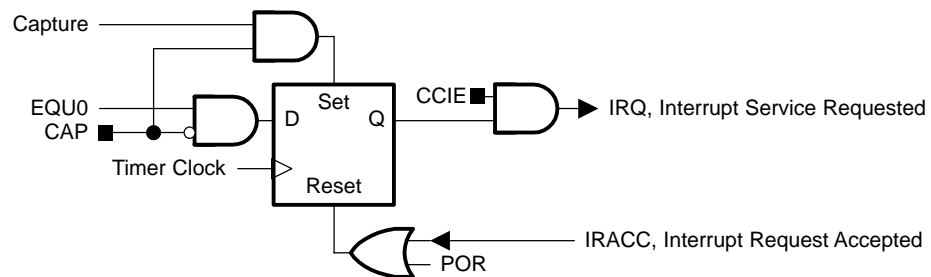
- TACCR0 interrupt vector for TACCR0 CCIFG
- TAIV interrupt vector for all other CCIFG flags and TAIFG

In capture mode any CCIFG flag is set when a timer value is captured in the associated TACCRx register. In compare mode, any CCIFG flag is set if TAR *counts* to the associated TACCRx value. Software may also set or clear any CCIFG flag. All CCIFG flags request an interrupt when their corresponding CCIE bit and the GIE bit are set.

### TACCR0 Interrupt

The TACCR0 CCIFG flag has the highest Timer\_A interrupt priority and has a dedicated interrupt vector as shown in Figure 12–15. The TACCR0 CCIFG flag is automatically reset when the TACCR0 interrupt request is serviced.

Figure 12–15. Capture/Compare TACCR0 Interrupt Flag



### TAIV, Interrupt Vector Generator

The TACCR1 CCIFG, TACCR2 CCIFG, and TAIFG flags are prioritized and combined to source a single interrupt vector. The interrupt vector register TAIV is used to determine which flag requested an interrupt.

The highest priority enabled interrupt generates a number in the TAIV register (see register description). This number can be evaluated or added to the program counter to automatically enter the appropriate software routine. Disabled Timer\_A interrupts do not affect the TAIV value.

Any access, read or write, of the TAIV register automatically resets the highest pending interrupt flag. If another interrupt flag is set, another interrupt is immediately generated after servicing the initial interrupt. For example, if the TACCR1 and TACCR2 CCIFG flags are set when the interrupt service routine accesses the TAIV register, TACCR1 CCIFG is reset automatically. After the RETI instruction of the interrupt service routine is executed, the TACCR2 CCIFG flag will generate another interrupt.

**TAIV Software Example**

The following software example shows the recommended use of TAIV and the handling overhead. The TAIV value is added to the PC to automatically jump to the appropriate routine.

The numbers at the right margin show the necessary CPU cycles for each instruction. The software overhead for different interrupt sources includes interrupt latency and return-from-interrupt cycles, but not the task handling itself. The latencies are:

<input type="checkbox"/>	Capture/compare block TACCR0	11 cycles
<input type="checkbox"/>	Capture/compare blocks TACCR1, TACCR2	16 cycles
<input type="checkbox"/>	Timer overflow TAIFG	14 cycles

```

; Interrupt handler for TACCR0 CCIFG.                                Cycles
CCIFG_0_HND
;          ...          ; Start of handler Interrupt latency 6
          RETI          ;                                     5

; Interrupt handler for TAIFG, TACCR1 and TACCR2 CCIFG.

TA_HND    ...          ; Interrupt latency                    6
          ADD    &TAIV,PC ; Add offset to Jump table        3
          RETI          ; Vector 0: No interrupt             5
          JMP    CCIFG_1_HND ; Vector 2: TACCR1              2
          JMP    CCIFG_2_HND ; Vector 4: TACCR2              2
          RETI          ; Vector 6: Reserved                 5
          RETI          ; Vector 8: Reserved                 5

TAIFG_HND ...          ; Vector 10: TAIFG Flag
          ...          ; Task starts here
          RETI          ;                                     5

CCIFG_2_HND ...          ; Vector 4: TACCR2
          ...          ; Task starts here
          RETI          ; Back to main program                5

CCIFG_1_HND ...          ; Vector 2: TACCR1
          ...          ; Task starts here
          RETI          ; Back to main program                5

```

## 12.3 Timer\_A Registers

The Timer\_A registers are listed in Table 12–3 and Table 12–4.

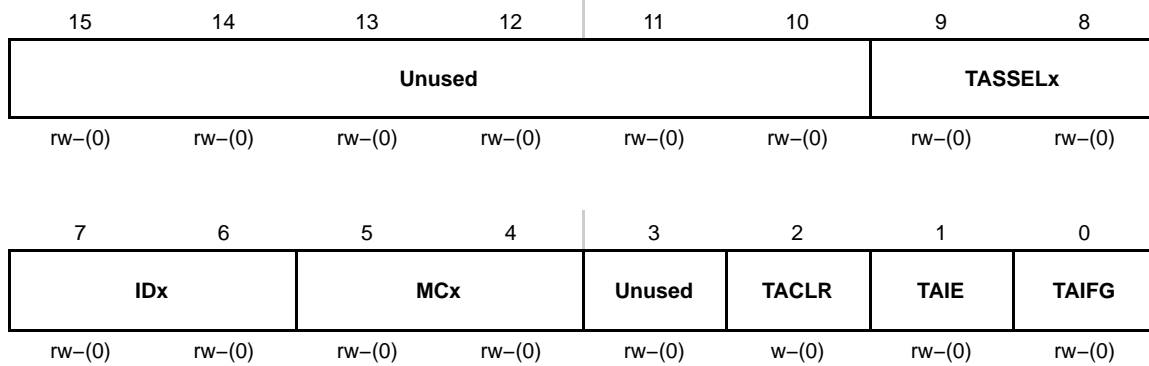
Table 12–3. Timer\_A3 Registers

Register	Short Form	Register Type	Address	Initial State
Timer_A control Timer0_A3 Control	TACTL/ TA0CTL	Read/write	0160h	Reset with POR
Timer_A counter Timer0_A3 counter	TAR/ TA0R	Read/write	0170h	Reset with POR
Timer_A capture/compare control 0 Timer0_A3 capture/compare control 0	TACCTL0/ TA0CCTL	Read/write	0162h	Reset with POR
Timer_A capture/compare 0 Timer0_A3 capture/compare 0	TACCR0/ TA0CCR0	Read/write	0172h	Reset with POR
Timer_A capture/compare control 1 Timer0_A3 capture/compare control 1	TACCTL1/ TA0CCTL1	Read/write	0164h	Reset with POR
Timer_A capture/compare 1 Timer0_A3 capture/compare 1	TACCR1/ TA0CCR1	Read/write	0174h	Reset with POR
Timer_A capture/compare control 2 Timer0_A3 capture/compare control 2	TACCTL2/ TA0CCTL2	Read/write	0166h	Reset with POR
Timer_A capture/compare 2 Timer0_A3 capture/compare 2	TACCR2/ TA0CCR2	Read/write	0176h	Reset with POR
Timer_A interrupt vector Timer0_A3 interrupt vector	TAIV/ TA0IV	Read only	012Eh	Reset with POR

Table 12–4. Timer1\_A5 Registers

Register	Short Form	Register Type	Address	Initial State
Timer1_A5 control	TA1CTL	Read/write	0180h	Reset with POR
Timer1_A5 counter	TA1R	Read/write	0190h	Reset with POR
Timer1_A5 capture/compare control 0	TA1CCTL0	Read/write	0182h	Reset with POR
Timer1_A5 capture/compare 0	TA1CCR0	Read/write	0192h	Reset with POR
Timer1_A5 capture/compare control 1	TA1CCTL1	Read/write	0184h	Reset with POR
Timer1_A5 capture/compare 1	TA1CCR1	Read/write	0194h	Reset with POR
Timer1_A5 capture/compare control 2	TA1CCTL2	Read/write	0186h	Reset with POR
Timer1_A5 capture/compare 2	TA1CCR2	Read/write	0196h	Reset with POR
Timer1_A5 capture/compare control 3	TA1CCTL3	Read/write	0188h	Reset with POR
Timer1_A5 capture/compare 3	TA1CCR3	Read/write	0198h	Reset with POR
Timer1_A5 capture/compare control 4	TA1CCTL4	Read/write	018Ah	Reset with POR
Timer1_A5 capture/compare 4	TA1CCR4	Read/write	019Ah	Reset with POR
Timer1_A5 Interrupt Vector	TA1IV	Read only	011Eh	Reset with POR

**TACTL, Timer\_A Control Register**



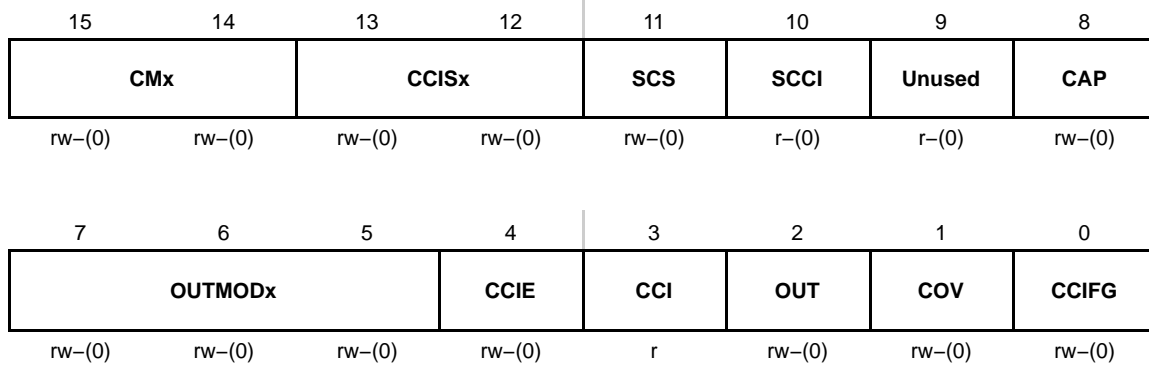
<b>Unused</b>	Bits 15-10	Unused
<b>TASSELx</b>	Bits 9-8	Timer_A clock source select 00 TACLK 01 ACLK 10 SMCLK 11 Inverted TACLK
<b>IDx</b>	Bits 7-6	Input divider. These bits select the divider for the input clock. 00 /1 01 /2 10 /4 11 /8
<b>MCx</b>	Bits 5-4	Mode control. Setting MCx = 00h when Timer_A is not in use conserves power. 00 Stop mode: the timer is halted 01 Up mode: the timer counts up to TACCR0 10 Continuous mode: the timer counts up to 0FFFFh 11 Up/down mode: the timer counts up to TACCR0 then down to 0000h
<b>Unused</b>	Bit 3	Unused
<b>TACLRL</b>	Bit 2	Timer_A clear. Setting this bit resets TAR, the TACLK divider, and the count direction. The TACLRL bit is automatically reset and is always read as zero.
<b>TAIE</b>	Bit 1	Timer_A interrupt enable. This bit enables the TAIFG interrupt request. 0 Interrupt disabled 1 Interrupt enabled
<b>TAIFG</b>	Bit 0	Timer_A interrupt flag 0 No interrupt pending 1 Interrupt pending

**TAR, Timer\_A Register**



**TARx**      Bits      Timer\_A register. The TAR register is the count of Timer\_A.  
                  15-0

**TACCTLx, Capture/Compare Control Register**



- CMx**      Bit      Capture mode

15-14    00    No capture

          01    Capture on rising edge

          10    Capture on falling edge

          11    Capture on both rising and falling edges
- CCISx**    Bit      Capture/compare input select. These bits select the TACCRx input signal. See the device-specific datasheet for specific signal connections.

13-12    00    CCIxA

          01    CCIxB

          10    GND

          11    V<sub>CC</sub>
- SCS**      Bit 11    Synchronize capture source. This bit is used to synchronize the capture input signal with the timer clock.

          0    Asynchronous capture

          1    Synchronous capture
- SCCI**     Bit 10    Synchronized capture/compare input. The selected CCI input signal is latched with the EQUx signal and can be read via this bit
- Unused**   Bit 9      Unused. Read only. Always read as 0.
- CAP**      Bit 8      Capture mode

          0    Compare mode

          1    Capture mode
- OUTMODx** Bits      Output mode. Modes 2, 3, 6, and 7 are not useful for TACCR0 because EQUx = EQU0.

7-5        000    OUT bit value

          001    Set

          010    Toggle/reset

          011    Set/reset

          100    Toggle

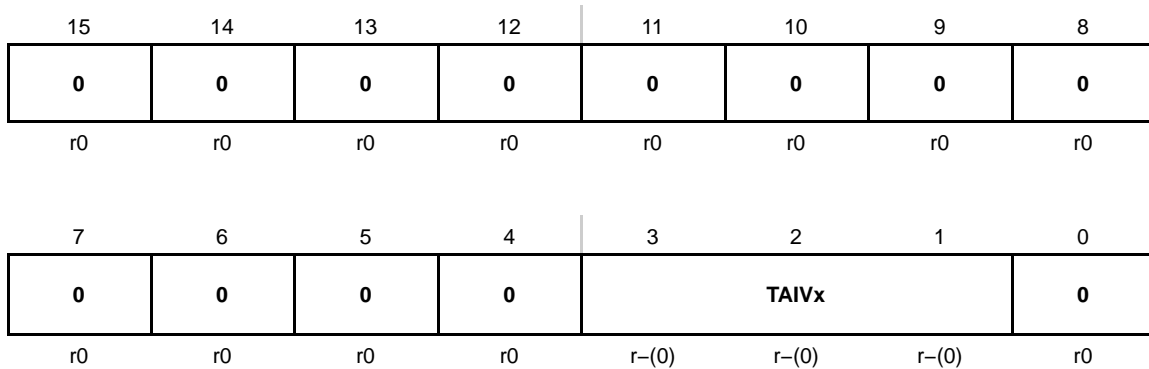
          101    Reset

          110    Toggle/set

          111    Reset/set

<b>CCIE</b>	Bit 4	Capture/compare interrupt enable. This bit enables the interrupt request of the corresponding CCIFG flag. 0 Interrupt disabled 1 Interrupt enabled
<b>CCI</b>	Bit 3	Capture/compare input. The selected input signal can be read by this bit.
<b>OUT</b>	Bit 2	Output. For output mode 0, this bit directly controls the state of the output. 0 Output low 1 Output high
<b>COV</b>	Bit 1	Capture overflow. This bit indicates a capture overflow occurred. COV must be reset with software. 0 No capture overflow occurred 1 Capture overflow occurred
<b>CCIFG</b>	Bit 0	Capture/compare interrupt flag 0 No interrupt pending 1 Interrupt pending

**TAIV, Timer\_A Interrupt Vector Register**



**TAIVx** Bits 15-0 Timer\_A Interrupt Vector value

TAIV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
00h	No interrupt pending	–	
02h	Capture/compare 1	TACCR1 CCIFG	Highest
04h	Capture/compare 2	TACCR2 CCIFG	
06h	Capture/compare 3 <sup>†</sup>	TACCR3 CCIFG	
08h	Capture/compare 4 <sup>†</sup>	TACCR4 CCIFG	
0Ah	Timer overflow	TAIFG	
0Ch	Reserved	–	
0Eh	Reserved	–	Lowest

<sup>†</sup> Timer1\_A5 only



**Timer\_B**

---

---

---

---

Timer\_B is a 16-bit timer/counter with multiple capture/compare registers. This chapter describes Timer\_B. Timer\_B3 (three capture/compare registers) is implemented in MSP430x43x devices. Timer\_B7 (seven capture/compare registers) is implemented in MSP430x44x.

<b>Topic</b>	<b>Page</b>
13.1 Timer_B Introduction .....	13-2
13.2 Timer_B Operation .....	13-4
13.3 Timer_B Registers .....	13-20

## 13.1 Timer\_B Introduction

Timer\_B is a 16-bit timer/counter with three or seven capture/compare registers. Timer\_B can support multiple capture/compares, PWM outputs, and interval timing. Timer\_B also has extensive interrupt capabilities. Interrupts may be generated from the counter on overflow conditions and from each of the capture/compare registers.

Timer\_B features include :

- Asynchronous 16-bit timer/counter with four operating modes and four selectable lengths
- Selectable and configurable clock source
- Three or seven configurable capture/compare registers
- Configurable outputs with PWM capability
- Double-buffered compare latches with synchronized loading
- Interrupt vector register for fast decoding of all Timer\_B interrupts

The block diagram of Timer\_B is shown in Figure 13–1.

---

**Note: Use of the Word *Count***

*Count* is used throughout this chapter. It means the counter must be in the process of counting for the action to take place. If a particular value is directly written to the counter, then an associated action does not take place.

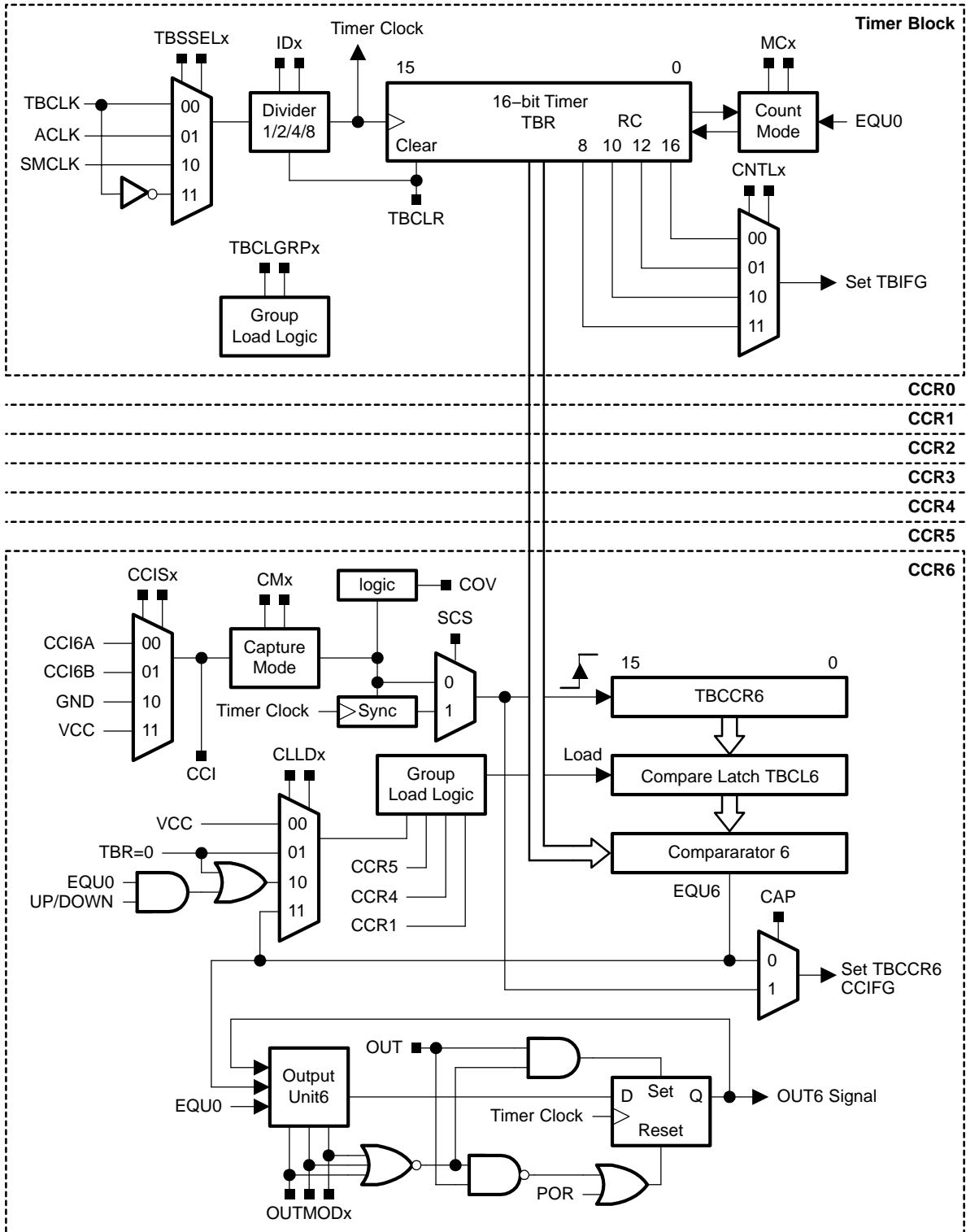
---

### 13.1.1 Similarities and Differences From Timer\_A

Timer\_B is identical to Timer\_A with the following exceptions:

- The length of Timer\_B is programmable to be 8, 10, 12, or 16 bits.
- Timer\_B TBCCR<sub>x</sub> registers are double-buffered and can be grouped.
- All Timer\_B outputs can be put into a high-impedance state.
- The SCCI bit function is not implemented in Timer\_B.

Figure 13-1. Timer\_B Block Diagram



## 13.2 Timer\_B Operation

The Timer\_B module is configured with user software. The setup and operation of Timer\_B is discussed in the following sections.

### 13.2.1 16-Bit Timer Counter

The 16-bit timer/counter register, TBR, increments or decrements (depending on mode of operation) with each rising edge of the clock signal. TBR can be read or written with software. Additionally, the timer can generate an interrupt when it overflows.

TBR may be cleared by setting the TBCLR bit. Setting TBCLR also clears the clock divider and count direction for up/down mode.

---

**Note: Modifying Timer\_B Registers**

It is recommended to stop the timer before modifying its operation (with exception of the interrupt enable, interrupt flag, and TBCLR) to avoid errant operating conditions.

When the TBCLK is asynchronous to the CPU clock, any read from TBR should occur while the timer is not operating or the results may be unpredictable. Alternatively, the timer may be read multiple times while operating, and a majority vote taken in software to determine the correct reading. Any write to TBR will take effect immediately.

---

### TBR Length

Timer\_B is configurable to operate as an 8-, 10-, 12-, or 16-bit timer with the CNTLx bits. The maximum count value,  $TBR_{(max)}$ , for the selectable lengths is 0FFh, 03FFh, 0FFFh, and 0FFFFh, respectively. Data written to the TBR register in 8-, 10-, and 12-bit mode is right-justified with leading zeros.

### Clock Source Select and Divider

The timer clock TBCLK can be sourced from ACLK, SMCLK, or externally via TBCLK or INCLK. The clock source is selected with the TBSSELx bits. The selected clock source may be passed directly to the timer or divided by 2, 4, or 8, using the IDx bits. The TBCLK divider is reset when TBCLR is set.

### 13.2.2 Starting the Timer

The timer may be started or restarted in the following ways:

- The timer counts when  $MCx > 0$  and the clock source is active.
- When the timer mode is either up or up/down, the timer may be stopped by loading 0 to TBCL0. The timer may then be restarted by loading a nonzero value to TBCL0. In this scenario, the timer starts incrementing in the up direction from zero.

### 13.2.3 Timer Mode Control

The timer has four modes of operation as described in Table 13–1: stop, up, continuous, and up/down. The operating mode is selected with the MCx bits.

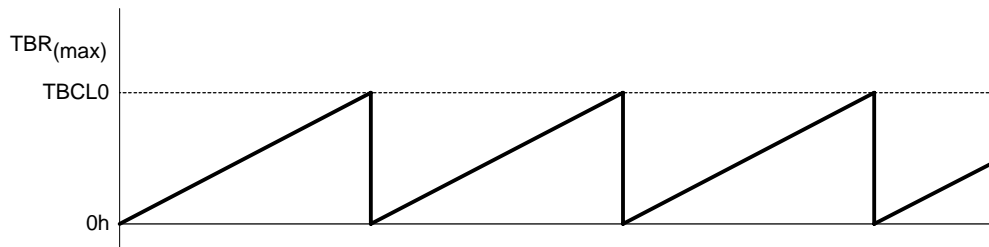
Table 13–1. Timer Modes

MCx	Mode	Description
00	Stop	The timer is halted.
01	Up	The timer repeatedly counts from zero to the value of compare register TBCL0.
10	Continuous	The timer repeatedly counts from zero to the value selected by the TBCNTLx bits.
11	Up/down	The timer repeatedly counts from zero up to the value of TBCL0 and then back down to zero.

## Up Mode

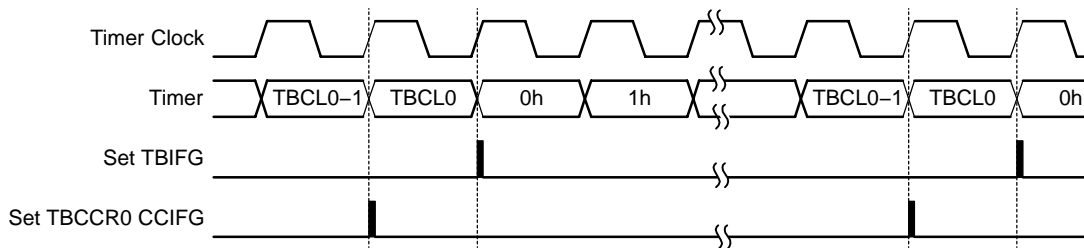
The up mode is used if the timer period must be different from  $TBR_{(max)}$  counts. The timer repeatedly counts up to the value of compare latch TBCL0, which defines the period, as shown in Figure 13–2. The number of timer counts in the period is TBCL0+1. When the timer value equals TBCL0 the timer restarts counting from zero. If up mode is selected when the timer value is greater than TBCL0, the timer immediately restarts counting from zero.

Figure 13–2. Up Mode



The TBCCR0 CCIFG interrupt flag is set when the timer *counts* to the TBCL0 value. The TBIFG interrupt flag is set when the timer *counts* from TBCL0 to zero. Figure 12–3 shows the flag set cycle.

Figure 13–3. Up Mode Flag Setting



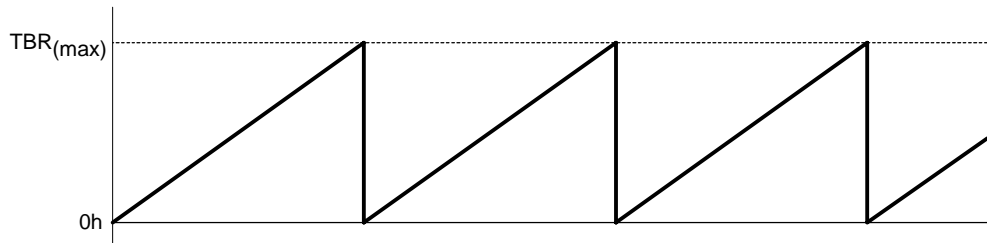
## Changing the Period Register TBCL0

When changing TBCL0 while the timer is running and when the TBCL0 load mode is *immediate*, if the new period is greater than or equal to the old period, or greater than the current count value, the timer counts up to the new period. If the new period is less than the current count value, the timer rolls to zero. However, one additional count may occur before the counter rolls to zero.

### Continuous Mode

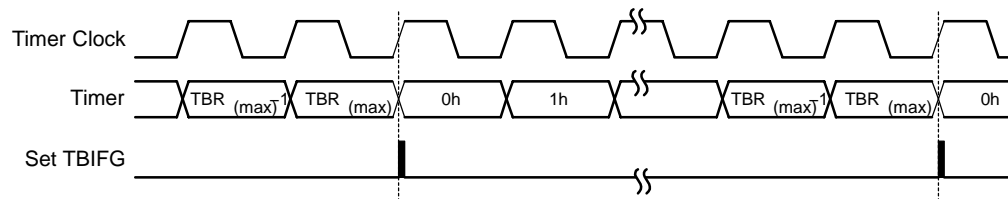
In continuous mode the timer repeatedly counts up to  $TBR_{(max)}$  and restarts from zero as shown in Figure 13–4. The compare latch TBCL0 works the same way as the other capture/compare registers.

Figure 13–4. Continuous Mode



The TBIFG interrupt flag is set when the timer counts from  $TBR_{(max)}$  to zero. Figure 13–5 shows the flag set cycle.

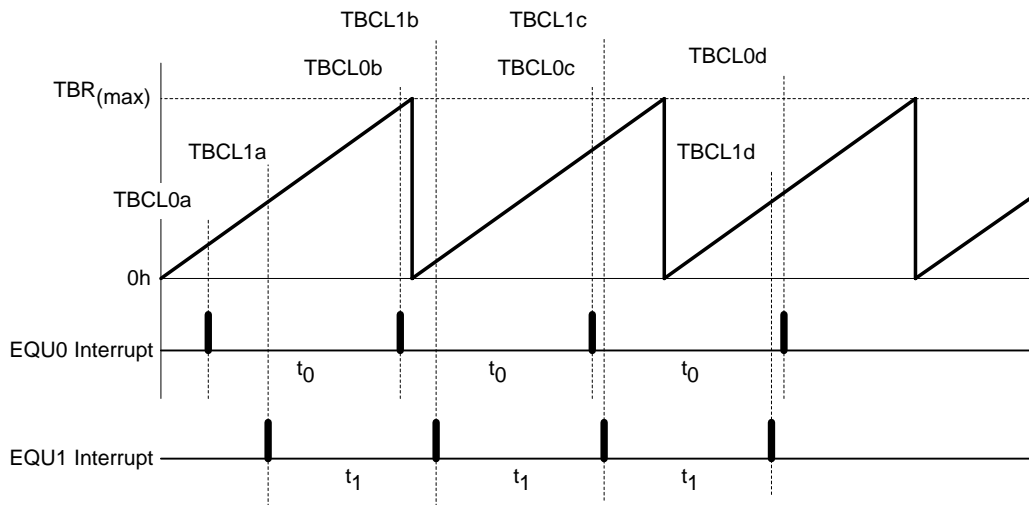
Figure 13–5. Continuous Mode Flag Setting



### Use of the Continuous Mode

The continuous mode can be used to generate independent time intervals and output frequencies. Each time an interval is completed, an interrupt is generated. The next time interval is added to the TBCLx latch in the interrupt service routine. Figure 13–6 shows two separate time intervals  $t_0$  and  $t_1$  being added to the capture/compare registers. The time interval is controlled by hardware, not software, without impact from interrupt latency. Up to three (Timer\_B3) or 7 (Timer\_B7) independent time intervals or output frequencies can be generated using capture/compare registers.

Figure 13–6. Continuous Mode Time Intervals



Time intervals can be produced with other modes as well, where TBCL0 is used as the period register. Their handling is more complex since the sum of the old TBCLx data and the new period can be higher than the TBCL0 value. When the sum of the previous TBCLx value plus  $t_x$  is greater than the TBCL0 data, the old TBCL0 value must be subtracted to obtain the correct time interval.



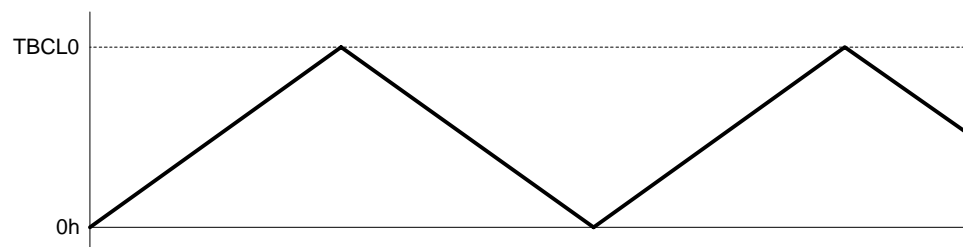
## Up/Down Mode

The up/down mode is used if the timer period must be different from  $TBR_{(max)}$  counts, and if symmetrical pulse generation is needed. The timer repeatedly counts up to the value of compare latch TBCL0, and back down to zero, as shown in Figure 13–7. The period is twice the value in TBCL0.

**Note: TBCL0 > TBR(max)**

If  $TBCL0 > TBR_{(max)}$ , the counter operates as if it were configured for continuous mode. It does not count down from  $TBR_{(max)}$  to zero.

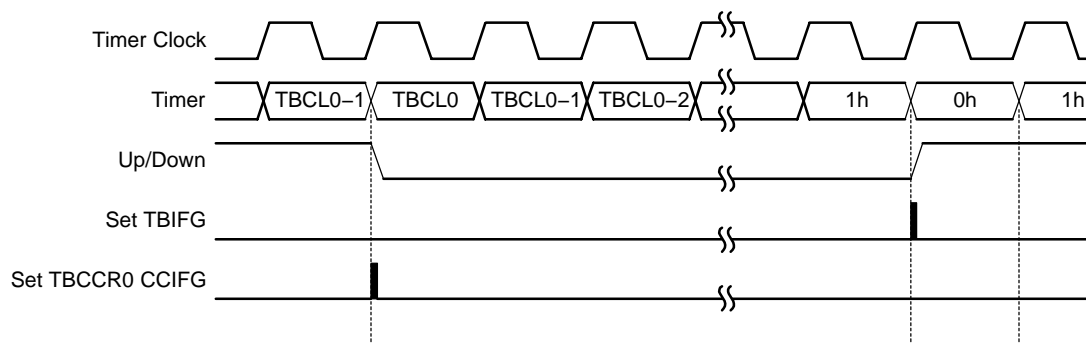
Figure 13–7. Up/Down Mode



The count direction is latched. This allows the timer to be stopped and then restarted in the same direction it was counting before it was stopped. If this is not desired, the TBCLR bit must be used to clear the direction. The TBCLR bit also clears the TBR value and the TBCLK divider.

In up/down mode, the TBCCR0 CCIFG interrupt flag and the TBIFG interrupt flag are set only once during the period, separated by 1/2 the timer period. The TBCCR0 CCIFG interrupt flag is set when the timer counts from  $TBCL0-1$  to  $TBCL0$ , and TBIFG is set when the timer completes counting down from  $0001h$  to  $0000h$ . Figure 13–8 shows the flag set cycle.

Figure 13–8. Up/Down Mode Flag Setting



### Changing the Value of Period Register TBCL0

When changing TBCL0 while the timer is running, and counting in the down direction, and when the TBCL0 load mode is *immediate*, the timer continues its descent until it reaches zero. The new period takes effect after the counter counts down to zero.

If the timer is counting in the up direction when the new period is latched into TBCL0, and the new period is greater than or equal to the old period, or greater than the current count value, the timer counts up to the new period before counting down. When the timer is counting in the up direction, and the new period is less than the current count value when TBCL0 is loaded, the timer begins counting down. However, one additional count may occur before the counter begins counting down.

### Use of the Up/Down Mode

The up/down mode supports applications that require dead times between output signals (see section *Timer\_B Output Unit*). For example, to avoid overload conditions, two outputs driving an H-bridge must never be in a high state simultaneously. In the example shown in Figure 13–9 the  $t_{dead}$  is:

$$t_{dead} = t_{timer} \times (TBCL1 - TBCL3)$$

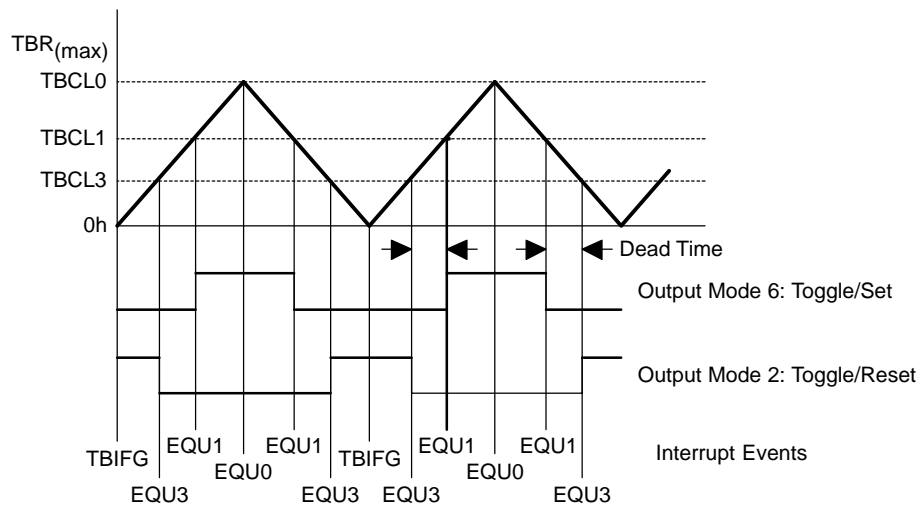
With:  $t_{dead}$  Time during which both outputs need to be inactive

$t_{timer}$  Cycle time of the timer clock

TBCLx Content of compare latch x

The ability to simultaneously load grouped compare latches assures the dead times.

Figure 13–9. Output Unit in Up/Down Mode



### 13.2.4 Capture/Compare Blocks

Three or seven identical capture/compare blocks, TBCCR<sub>x</sub>, are present in Timer\_B. Any of the blocks may be used to capture the timer data or to generate time intervals.

#### Capture Mode

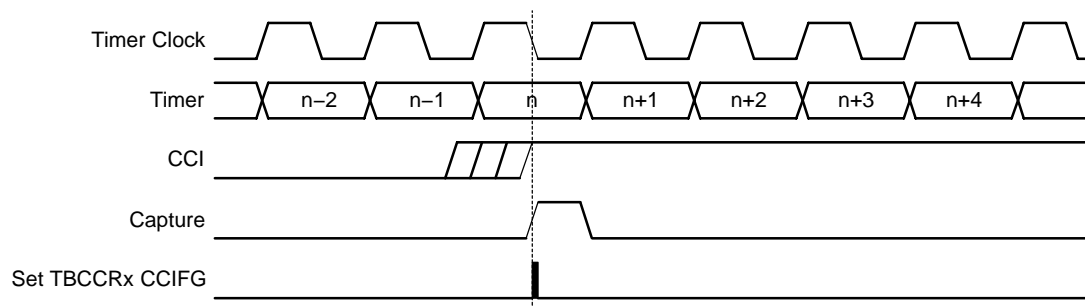
The capture mode is selected when CAP = 1. Capture mode is used to record time events. It can be used for speed computations or time measurements. The capture inputs CCI<sub>x</sub>A and CCI<sub>x</sub>B are connected to external pins or internal signals and are selected with the CCIS<sub>x</sub> bits. The CM<sub>x</sub> bits select the capture edge of the input signal as rising, falling, or both. A capture occurs on the selected edge of the input signal. If a capture is performed:

- The timer value is copied into the TBCCR<sub>x</sub> register
- The interrupt flag CCIFG is set

The input signal level can be read at any time via the CCI bit. MSP430x4xx family devices may have different signals connected to CCI<sub>x</sub>A and CCI<sub>x</sub>B. Refer to the device-specific datasheet for the connections of these signals.

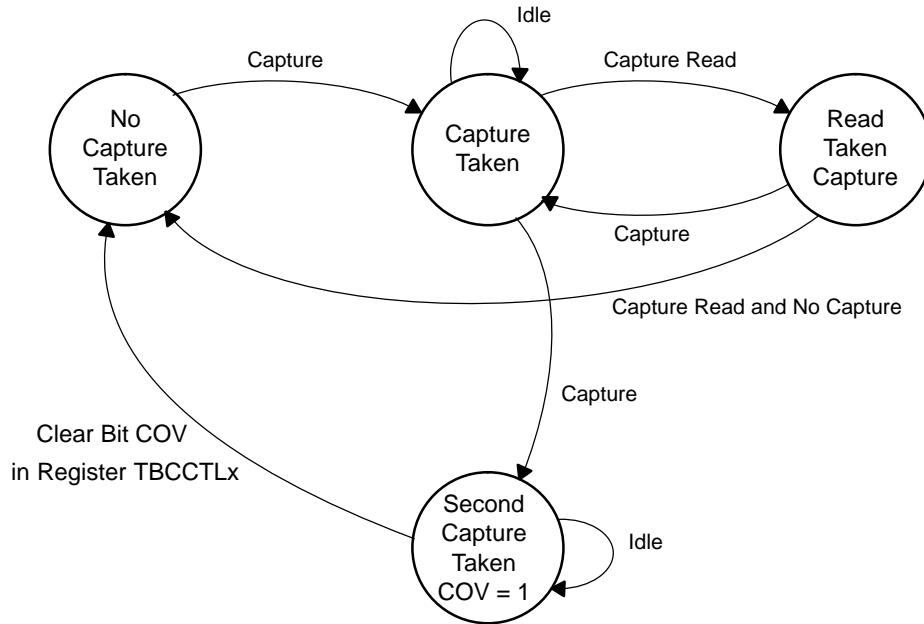
The capture signal can be asynchronous to the timer clock and cause a race condition. Setting the SCS bit will synchronize the capture with the next timer clock. Setting the SCS bit to synchronize the capture signal with the timer clock is recommended. This is illustrated in Figure 13–10.

Figure 13–10. Capture Signal (SCS=1)



Overflow logic is provided in each capture/compare register to indicate if a second capture was performed before the value from the first capture was read. Bit COV is set when this occurs as shown in Figure 13–11. COV must be reset with software.

Figure 13–11. Capture Cycle



**Capture Initiated by Software**

Captures can be initiated by software. The CMx bits can be set for capture on both edges. Software then sets bit CCIS1=1 and toggles bit CCIS0 to switch the capture signal between V<sub>CC</sub> and GND, initiating a capture each time CCIS0 changes state:

```

MOV    #CAP+SCS+CCIS1+CM_3,&TBCCTLx ; Setup TBCCTLx
XOR    #CCIS0,&TBCCTLx                ; TBCCTLx = TBR
  
```

**Compare Mode**

The compare mode is selected when CAP = 0. Compare mode is used to generate PWM output signals or interrupts at specific time intervals. When TBR counts to the value in a TBCLx:

- Interrupt flag CCIFG is set
- Internal signal EQUx = 1
- EQUx affects the output according to the output mode

### Compare Latch TBCLx

The TBCCR<sub>x</sub> compare latch, TBCL<sub>x</sub>, holds the data for the comparison to the timer value in compare mode. TBCL<sub>x</sub> is buffered by TBCCR<sub>x</sub>. The buffered compare latch gives the user control over when a compare period updates. The user cannot directly access TBCL<sub>x</sub>. Compare data is written to each TBCCR<sub>x</sub> and automatically transferred to TBCL<sub>x</sub>. The timing of the transfer from TBCCR<sub>x</sub> to TBCL<sub>x</sub> is user-selectable with the CLLD<sub>x</sub> bits as described in Table 13–2.

Table 13–2. TBCL<sub>x</sub> Load Events

CLLD <sub>x</sub>	Description
00	New data is transferred from TBCCR <sub>x</sub> to TBCL <sub>x</sub> immediately when TBCCR <sub>x</sub> is written to.
01	New data is transferred from TBCCR <sub>x</sub> to TBCL <sub>x</sub> when TBR <i>counts</i> to 0
10	New data is transferred from TBCCR <sub>x</sub> to TBCL <sub>x</sub> when TBR <i>counts</i> to 0 for up and continuous modes. New data is transferred to from TBCCR <sub>x</sub> to TBCL <sub>x</sub> when TBR <i>counts</i> to the old TBCL <sub>0</sub> value or to 0 for up/down mode
11	New data is transferred from TBCCR <sub>x</sub> to TBCL <sub>x</sub> when TBR <i>counts</i> to the old TBCL <sub>x</sub> value.

### Grouping Compare Latches

Multiple compare latches may be grouped together for simultaneous updates with the TBCLGRP<sub>x</sub> bits. When using groups, the CLLD<sub>x</sub> bits of the lowest numbered TBCCR<sub>x</sub> in the group determine the load event for each compare latch of the group, except when TBCLGRP = 3, as shown in Table 13–3. The CLLD<sub>x</sub> bits of the controlling TBCCR<sub>x</sub> must not be set to zero. When the CLLD<sub>x</sub> bits of the controlling TBCCR<sub>x</sub> are set to zero, all compare latches update immediately when their corresponding TBCCR<sub>x</sub> is written - no compare latches are grouped.

Two conditions must exist for the compare latches to be loaded when grouped. First, all TBCCR<sub>x</sub> registers of the group must be updated, even when new TBCCR<sub>x</sub> data = old TBCCR<sub>x</sub> data. Second, the load event must occur.

Table 13–3. Compare Latch Operating Modes

TBCLGRP <sub>x</sub>	Grouping	Update Control
00	None	Individual
01	TBCL1+TBCL2 TBCL3+TBCL4 TBCL5+TBCL6	TBCCR1 TBCCR3 TBCCR5
10	TBCL1+TBCL2+TBCL3 TBCL4+TBCL5+TBCL6	TBCCR1 TBCCR4
11	TBCL0+TBCL1+TBCL2+ TBCL3+TBCL4+TBCL5+TBCL6	TBCCR1

### 13.2.5 Output Unit

Each capture/compare block contains an output unit. The output unit is used to generate output signals such as PWM signals. Each output unit has eight operating modes that generate signals based on the EQU0 and EQUx signals. The TBOUTH pin function can be used to put all Timer\_B outputs into a high-impedance state. When the TBOUTH pin function is selected for the pin, and when the pin is pulled high, all Timer\_B outputs are in a high-impedance state.

### Output Modes

The output modes are defined by the OUTMODx bits and are described in Table 13–4. The OUTx signal is changed with the rising edge of the timer clock for all modes except mode 0. Output modes 2, 3, 6, and 7 are not useful for output unit 0 because EQUx = EQU0.

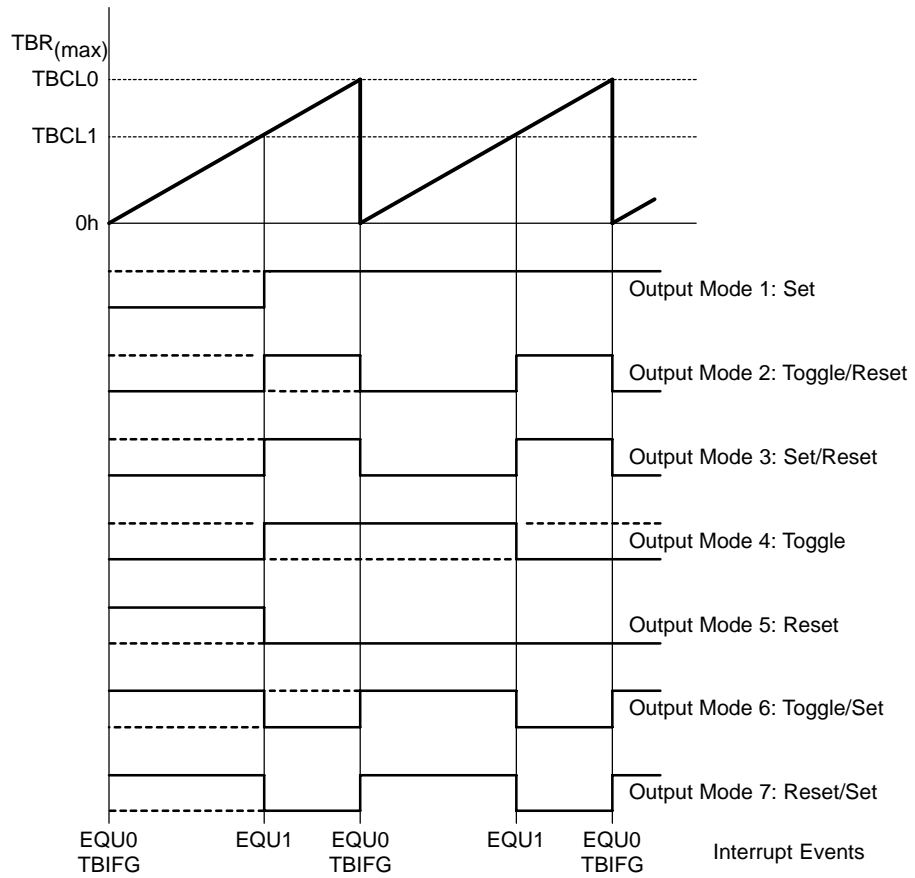
Table 13–4. Output Modes

OUTMODx	Mode	Description
000	Output	The output signal OUTx is defined by the OUTx bit. The OUTx signal updates immediately when OUTx is updated.
001	Set	The output is set when the timer <i>counts</i> to the TBCLx value. It remains set until a reset of the timer, or until another output mode is selected and affects the output.
010	Toggle/Reset	The output is toggled when the timer <i>counts</i> to the TBCLx value. It is reset when the timer <i>counts</i> to the TBCL0 value.
011	Set/Reset	The output is set when the timer <i>counts</i> to the TBCLx value. It is reset when the timer <i>counts</i> to the TBCL0 value.
100	Toggle	The output is toggled when the timer <i>counts</i> to the TBCLx value. The output period is double the timer period.
101	Reset	The output is reset when the timer <i>counts</i> to the TBCLx value. It remains reset until another output mode is selected and affects the output.
110	Toggle/Set	The output is toggled when the timer <i>counts</i> to the TBCLx value. It is set when the timer <i>counts</i> to the TBCL0 value.
111	Reset/Set	The output is reset when the timer <i>counts</i> to the TBCLx value. It is set when the timer <i>counts</i> to the TBCL0 value.

**Output Example—Timer in Up Mode**

The OUTx signal is changed when the timer *counts* up to the TBCLx value, and rolls from TBCL0 to zero, depending on the output mode. An example is shown in Figure 13–12 using TBCL0 and TBCL1.

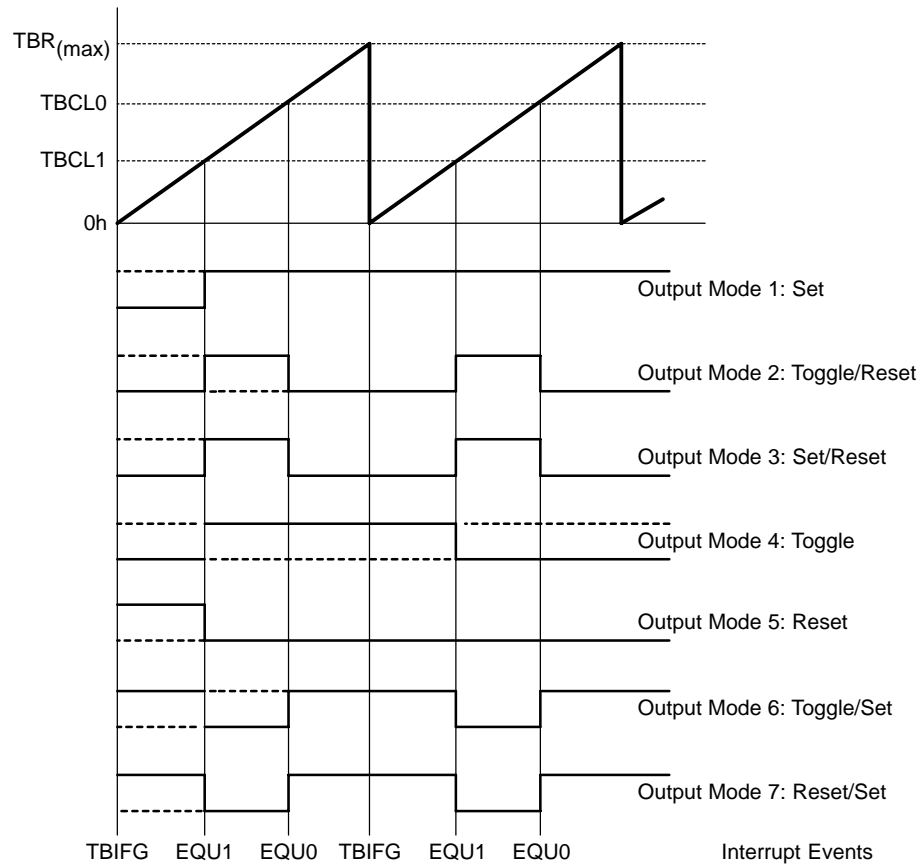
Figure 13–12. Output Example—Timer in Up Mode



**Output Example—Timer in Continuous Mode**

The OUTx signal is changed when the timer reaches the TBCLx and TBCL0 values, depending on the output mode. An example is shown in Figure 13–13 using TBCL0 and TBCL1.

Figure 13–13. Output Example—Timer in Continuous Mode

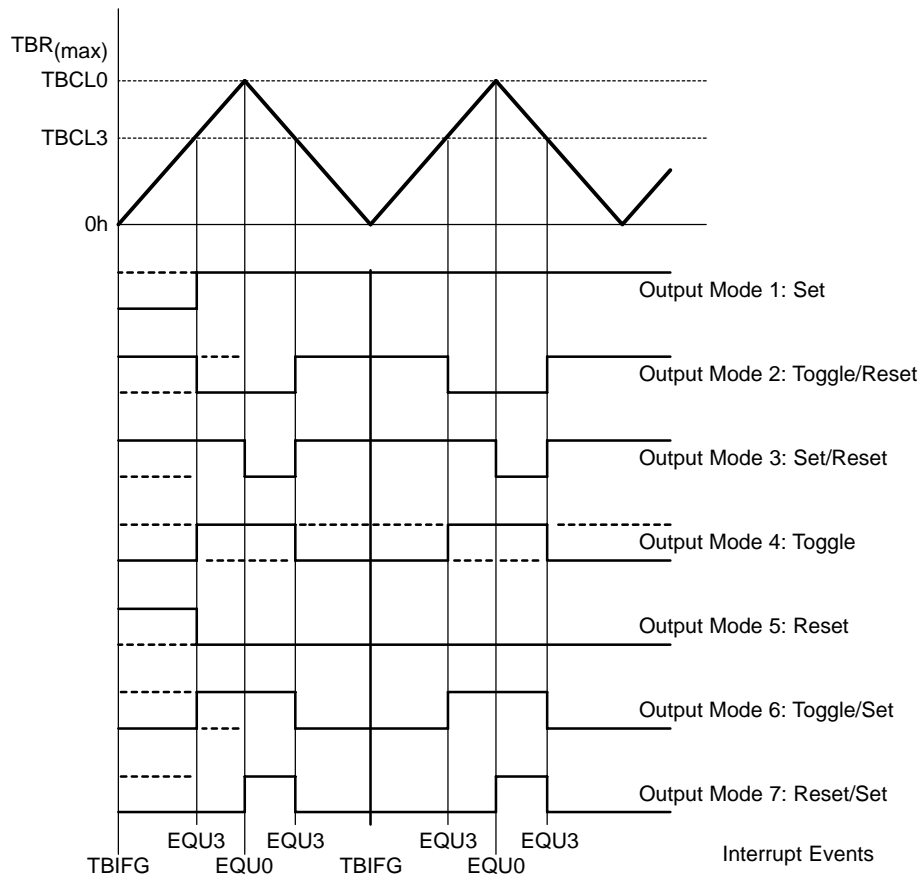




**Output Example – Timer in Up/Down Mode**

The OUTx signal changes when the timer equals TBCLx in either count direction and when the timer equals TBCL0, depending on the output mode. An example is shown in Figure 13–14 using TBCL0 and TBCL3.

Figure 13–14. Output Example—Timer in Up/Down Mode



**Note: Switching Between Output Modes**

When switching between output modes, one of the OUTMODx bits should remain set during the transition, unless switching to mode 0. Otherwise, output glitching can occur because a NOR gate decodes output mode 0. A safe method for switching between output modes is to use output mode 7 as a transition state:

```
BIS #OUTMOD_7,&TBCCTLx ; Set output mode=7
BIC #OUTMODx,&TBCCTLx ; Clear unwanted bits
```

### 13.2.6 Timer\_B Interrupts

Two interrupt vectors are associated with the 16-bit Timer\_B module:

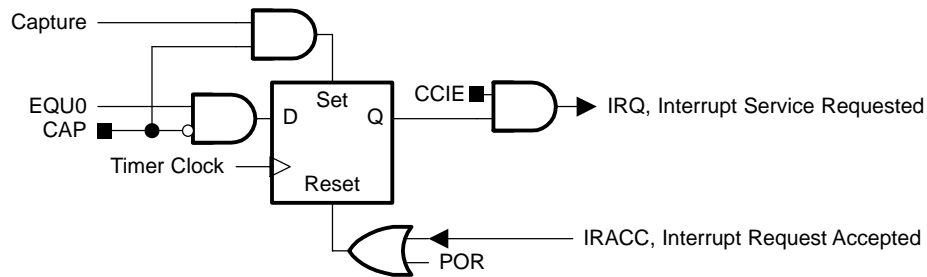
- TBCCR0 interrupt vector for TBCCR0 CCIFG
- TBIV interrupt vector for all other CCIFG flags and TBIFG

In capture mode, any CCIFG flag is set when a timer value is captured in the associated TBCCR<sub>x</sub> register. In compare mode, any CCIFG flag is set when TBR *counts* to the associated TBCL<sub>x</sub> value. Software may also set or clear any CCIFG flag. All CCIFG flags request an interrupt when their corresponding CCIE bit and the GIE bit are set.

#### TBCCR0 Interrupt Vector

The TBCCR0 CCIFG flag has the highest Timer\_B interrupt priority and has a dedicated interrupt vector as shown in Figure 13–15. The TBCCR0 CCIFG flag is automatically reset when the TBCCR0 interrupt request is serviced.

Figure 13–15. Capture/Compare TBCCR0 Interrupt Flag



#### TBIV, Interrupt Vector Generator

The TBIFG flag and TBCCR<sub>x</sub> CCIFG flags (excluding TBCCR0 CCIFG) are prioritized and combined to source a single interrupt vector. The interrupt vector register TBIV is used to determine which flag requested an interrupt.

The highest priority enabled interrupt (excluding TBCCR0 CCIFG) generates a number in the TBIV register (see register description). This number can be evaluated or added to the program counter to automatically enter the appropriate software routine. Disabled Timer\_B interrupts do not affect the TBIV value.

Any access, read or write, of the TBIV register automatically resets the highest pending interrupt flag. If another interrupt flag is set, another interrupt is immediately generated after servicing the initial interrupt. For example, if the TBCCR1 and TBCCR2 CCIFG flags are set when the interrupt service routine accesses the TBIV register, TBCCR1 CCIFG is reset automatically. After the RETI instruction of the interrupt service routine is executed, the TBCCR2 CCIFG flag will generate another interrupt.

## TBIV, Interrupt Handler Examples

The following software example shows the recommended use of TBIV and the handling overhead. The TBIV value is added to the PC to automatically jump to the appropriate routine.

The numbers at the right margin show the necessary CPU clock cycles for each instruction. The software overhead for different interrupt sources includes interrupt latency and return-from-interrupt cycles, but not the task handling itself. The latencies are:

<input type="checkbox"/> Capture/compare block CCR0	11 cycles
<input type="checkbox"/> Capture/compare blocks CCR1 to CCR6	16 cycles
<input type="checkbox"/> Timer overflow TBIFG	14 cycles

The following software example shows the recommended use of TBIV for Timer\_B3.

```

; Interrupt handler for TBCCR0 CCIFG.                                Cycles
CCIFG_0_HND
...           ; Start of handler Interrupt latency 6
RETI                                                5

; Interrupt handler for TBIFG, TBCCR1 and TBCCR2 CCIFG.
TB_HND  ...           ; Interrupt latency 6
ADD    &TBIV,PC    ; Add offset to Jump table 3
RETI   ...           ; Vector 0: No interrupt 5
JMP    CCIFG_1_HND ; Vector 2: Module 1 2
JMP    CCIFG_2_HND ; Vector 4: Module 2 2
RETI   ...           ; Vector 6
RETI   ...           ; Vector 8
RETI   ...           ; Vector 10
RETI   ...           ; Vector 12

TBIFG_HND ...           ; Vector 14: TIMOV Flag
...           ; Task starts here
RETI   ...           ; Back to main program 5

CCIFG_2_HND ...           ; Vector 4: Module 2
...           ; Task starts here
RETI   ...           ; Back to main program 5

; The Module 1 handler shows a way to look if any other
; interrupt is pending: 5 cycles have to be spent, but
; 9 cycles may be saved if another interrupt is pending
CCIFG_1_HND ...           ; Vector 6: Module 3
...           ; Task starts here
JMP    TB_HND     ; Look for pending ints 2

```

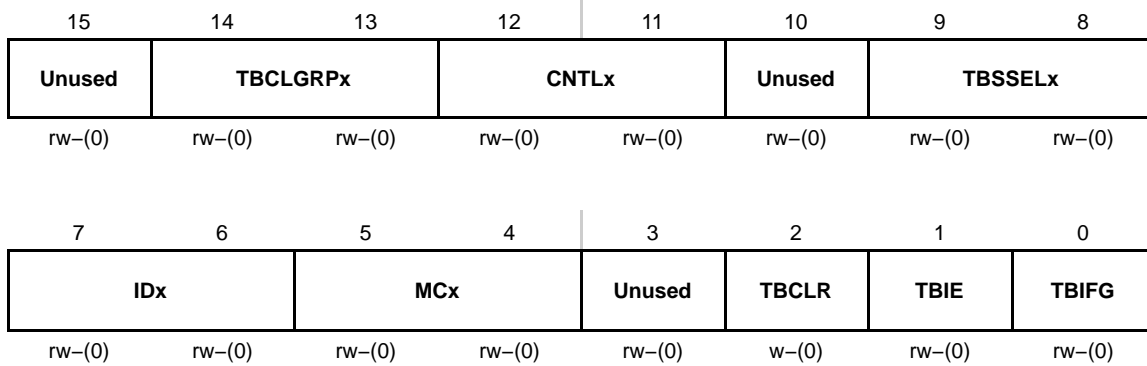
### 13.3 Timer\_B Registers

The Timer\_B registers are listed in Table 13–5.

Table 13–5. Timer\_B Registers

Register	Short Form	Register Type	Address	Initial State
Timer_B control	TBCTL	Read/write	0180h	Reset with POR
Timer_B counter	TBR	Read/write	0190h	Reset with POR
Timer_B capture/compare control 0	TBCCTL0	Read/write	0182h	Reset with POR
Timer_B capture/compare 0	TBCCR0	Read/write	0192h	Reset with POR
Timer_B capture/compare control 1	TBCCTL1	Read/write	0184h	Reset with POR
Timer_B capture/compare 1	TBCCR1	Read/write	0194h	Reset with POR
Timer_B capture/compare control 2	TBCCTL2	Read/write	0186h	Reset with POR
Timer_B capture/compare 2	TBCCR2	Read/write	0196h	Reset with POR
Timer_B capture/compare control 3	TBCCTL3	Read/write	0188h	Reset with POR
Timer_B capture/compare 3	TBCCR3	Read/write	0198h	Reset with POR
Timer_B capture/compare control 4	TBCCTL4	Read/write	018Ah	Reset with POR
Timer_B capture/compare 4	TBCCR4	Read/write	019Ah	Reset with POR
Timer_B capture/compare control 5	TBCCTL5	Read/write	018Ch	Reset with POR
Timer_B capture/compare 5	TBCCR5	Read/write	019Ch	Reset with POR
Timer_B capture/compare control 6	TBCCTL6	Read/write	018Eh	Reset with POR
Timer_B capture/compare 6	TBCCR6	Read/write	019Eh	Reset with POR
Timer_B Interrupt Vector	TBIV	Read only	011Eh	Reset with POR

**Timer\_B Control Register TBCTL**



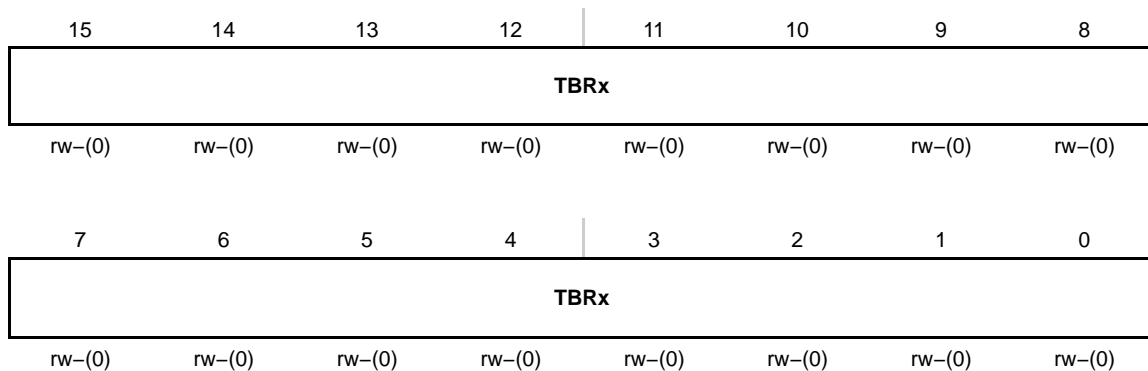
<b>Unused</b>	Bit 15	Unused
<b>TBCLGRP</b>	Bit 14-13	TBCL <sub>x</sub> group 00 Each TBCL <sub>x</sub> latch loads independently 01 TBCL <sub>1</sub> +TBCL <sub>2</sub> (TBCCR1 CLLD <sub>x</sub> bits control the update) TBCL <sub>3</sub> +TBCL <sub>4</sub> (TBCCR3 CLLD <sub>x</sub> bits control the update) TBCL <sub>5</sub> +TBCL <sub>6</sub> (TBCCR5 CLLD <sub>x</sub> bits control the update) TBCL <sub>0</sub> independent 10 TBCL <sub>1</sub> +TBCL <sub>2</sub> +TBCL <sub>3</sub> (TBCCR1 CLLD <sub>x</sub> bits control the update) TBCL <sub>4</sub> +TBCL <sub>5</sub> +TBCL <sub>6</sub> (TBCCR4 CLLD <sub>x</sub> bits control the update) TBCL <sub>0</sub> independent 11 TBCL <sub>0</sub> +TBCL <sub>1</sub> +TBCL <sub>2</sub> +TBCL <sub>3</sub> +TBCL <sub>4</sub> +TBCL <sub>5</sub> +TBCL <sub>6</sub> (TBCCR1 CLLD <sub>x</sub> bits control the update)
<b>CNTL<sub>x</sub></b>	Bits 12-11	Counter Length 00 16-bit, TBR <sub>(max)</sub> = 0FFFFh 01 12-bit, TBR <sub>(max)</sub> = 0FFFh 10 10-bit, TBR <sub>(max)</sub> = 03FFh 11 8-bit, TBR <sub>(max)</sub> = 0FFh
<b>Unused</b>	Bit 10	Unused
<b>TBSSEL<sub>x</sub></b>	Bits 9-8	Timer_B clock source select. 00 TBCLK 01 ACLK 10 SMCLK 11 Inverted TBCLK
<b>ID<sub>x</sub></b>	Bits 7-6	Input divider. These bits select the divider for the input clock. 00 /1 01 /2 10 /4 11 /8
<b>MC<sub>x</sub></b>	Bits 5-4	Mode control. Setting MC <sub>x</sub> = 00h when Timer_B is not in use conserves power. 00 Stop mode: the timer is halted 01 Up mode: the timer counts up to TBCL <sub>0</sub> 10 Continuous mode: the timer counts up to the value set by TBCNTL <sub>x</sub> 11 Up/down mode: the timer counts up to TBCL <sub>0</sub> and down to 0000h

## Timer\_B Registers

---

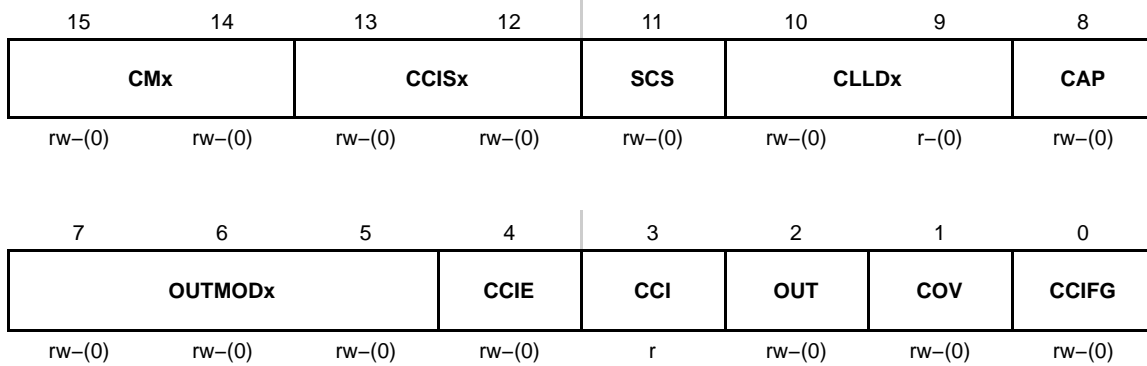
<b>Unused</b>	Bit 3	Unused
<b>TBCLR</b>	Bit 2	Timer_B clear. Setting this bit resets TBR, the TBCLK divider, and the count direction. The TBCLR bit is automatically reset and is always read as zero.
<b>TBIE</b>	Bit 1	Timer_B interrupt enable. This bit enables the TBIFG interrupt request. 0 Interrupt disabled 1 Interrupt enabled
<b>TBIFG</b>	Bit 0	Timer_B interrupt flag. 0 No interrupt pending 1 Interrupt pending

### TBR, Timer\_B Register



<b>TBRx</b>	Bits 15-0	Timer_B register. The TBR register is the count of Timer_B.
-------------	--------------	---

**TBCCTLx, Capture/Compare Control Register**



- CMx**      Bit      Capture mode

15-14    00    No capture

          01    Capture on rising edge

          10    Capture on falling edge

          11    Capture on both rising and falling edges
  
- CCISx**    Bit      Capture/compare input select. These bits select the TBCCR<sub>x</sub> input signal. See the device-specific datasheet for specific signal connections.

13-12    00    CCI<sub>x</sub>A

          01    CCI<sub>x</sub>B

          10    GND

          11    V<sub>CC</sub>
  
- SCS**      Bit 11    Synchronize capture source. This bit is used to synchronize the capture input signal with the timer clock.

          0    Asynchronous capture

          1    Synchronous capture
  
- CLLDx**    Bit      Compare latch load. These bits select the compare latch load event.

10-9     00    TBCL<sub>x</sub> loads on write to TBCCR<sub>x</sub>

          01    TBCL<sub>x</sub> loads when TBR *counts* to 0

          10    TBCL<sub>x</sub> loads when TBR *counts* to 0 (up or continuous mode)

                  TBCL<sub>x</sub> loads when TBR *counts* to TBCL<sub>0</sub> or to 0 (up/down mode)

          11    TBCL<sub>x</sub> loads when TBR *counts* to TBCL<sub>x</sub>
  
- CAP**      Bit 8     Capture mode

          0    Compare mode

          1    Capture mode
  
- OUTMODx** Bits     Output mode. Modes 2, 3, 6, and 7 are not useful for TBCL<sub>0</sub> because EQU<sub>x</sub> = EQU<sub>0</sub>.

7-5      000    OUT bit value

          001    Set

          010    Toggle/reset

          011    Set/reset

          100    Toggle

          101    Reset

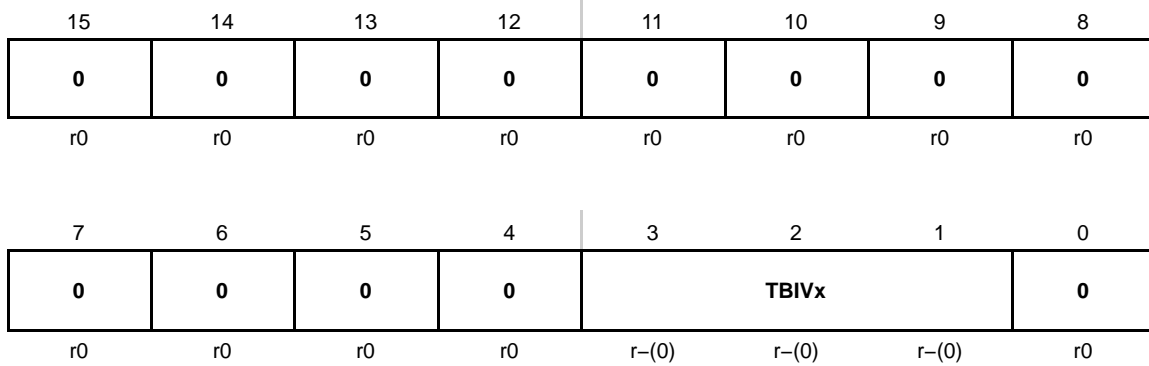
          110    Toggle/set

          111    Reset/set

<b>CCIE</b>	Bit 4	Capture/compare interrupt enable. This bit enables the interrupt request of the corresponding CCIFG flag. 0    Interrupt disabled 1    Interrupt enabled
<b>CCI</b>	Bit 3	Capture/compare input. The selected input signal can be read by this bit.
<b>OUT</b>	Bit 2	Output. For output mode 0, this bit directly controls the state of the output. 0    Output low 1    Output high
<b>COV</b>	Bit 1	Capture overflow. This bit indicates a capture overflow occurred. COV must be reset with software. 0    No capture overflow occurred 1    Capture overflow occurred
<b>CCIFG</b>	Bit 0	Capture/compare interrupt flag 0    No interrupt pending 1    Interrupt pending



**TBIV, Timer\_B Interrupt Vector Register**



**TBIVx**      Bits      Timer\_B interrupt vector value  
 15-0

TBIV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
00h	No interrupt pending	–	
02h	Capture/compare 1	TBCCR1 CCIFG	Highest
04h	Capture/compare 2	TBCCR2 CCIFG	
06h	Capture/compare 3 <sup>†</sup>	TBCCR3 CCIFG	
08h	Capture/compare 4 <sup>†</sup>	TBCCR4 CCIFG	
0Ah	Capture/compare 5 <sup>†</sup>	TBCCR5 CCIFG	
0Ch	Capture/compare 6 <sup>†</sup>	TBCCR6 CCIFG	
0Eh	Timer overflow	TBIFG	Lowest

<sup>†</sup> MSP430x4xx, devices only

# USART Peripheral Interface, UART Mode

---

---

---

The universal synchronous/asynchronous receive/transmit (USART) peripheral interface supports two serial modes with one hardware module. This chapter discusses the operation of the asynchronous UART mode. USART0 is implemented on the MSP430x42x and MSP430x43x devices. In addition to USART0, the MSP430x44x devices implement a second identical USART module, USART1.

<b>Topic</b>	<b>Page</b>
14.1 USART Introduction: UART Mode .....	14-2
14.2 USART Operation: UART Mode .....	14-4
14.3 USART Registers: UART Mode .....	14-21

## 14.1 USART Introduction: UART Mode

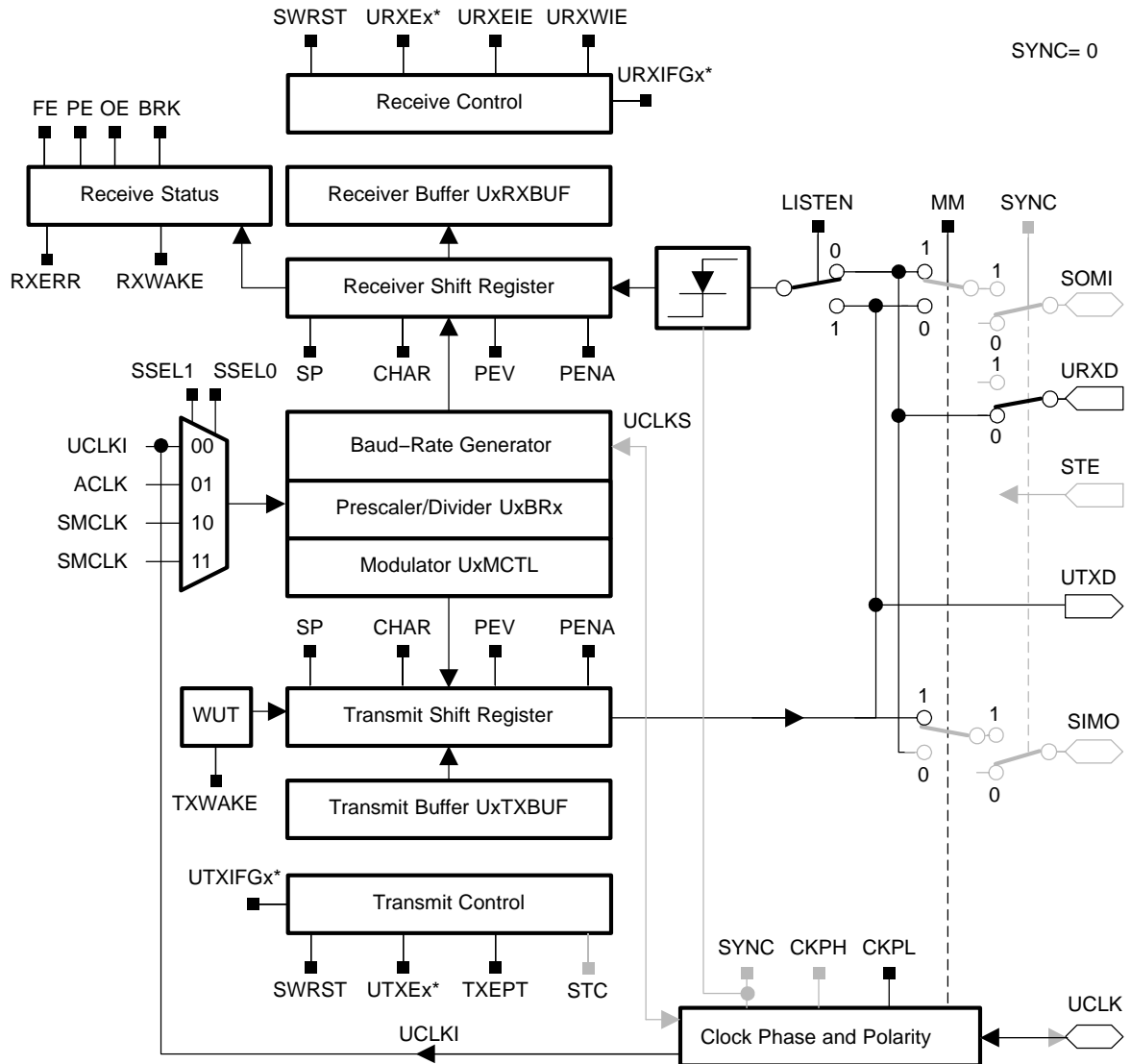
In asynchronous mode, the USART connects the MSP430 to an external system via two external pins, URXD and UTXD. UART mode is selected when the SYNC bit is cleared.

UART mode features include:

- 7- or 8-bit data with odd, even, or non-parity
- Independent transmit and receive shift registers
- Separate transmit and receive buffer registers
- LSB-first data transmit and receive
- Built-in idle-line and address-bit communication protocols for multiprocessor systems
- Receiver start-edge detection for auto-wake up from LPMx modes
- Programmable baud rate with modulation for fractional baud rate support
- Status flags for error detection and suppression and address detection
- Independent interrupt capability for receive and transmit

Figure 14–1 shows the USART when configured for UART mode.

Figure 14–1. USART Block Diagram: UART Mode



\* Refer to the device-specific datasheet for SFR locations

## 14.2 USART Operation: UART Mode

In UART mode, the USART transmits and receives characters at a bit rate asynchronous to another device. Timing for each character is based on the selected baud rate of the USART. The transmit and receive functions use the same baud rate frequency.

### 14.2.1 USART Initialization and Reset

The USART is reset by a PUC or by setting the SWRST bit. After a PUC, the SWRST bit is automatically set, keeping the USART in a reset condition. When set, the SWRST bit resets the URXIE<sub>x</sub>, UTXIE<sub>x</sub>, URXIFG<sub>x</sub>, RXWAKE, TXWAKE, RXERR, BRK, PE, OE, and FE bits and sets the UTXIFG<sub>x</sub> and TXEPT bits. The receive and transmit enable flags, URXEx and UTXEx, are not altered by SWRST. Clearing SWRST releases the USART for operation. See also chapter *USART Module, I2C mode* for USART0 when reconfiguring from I<sup>2</sup>C mode to UART mode.

**Note: Initializing or Re-Configuring the USART Module**

The required USART initialization/re-configuration process is:

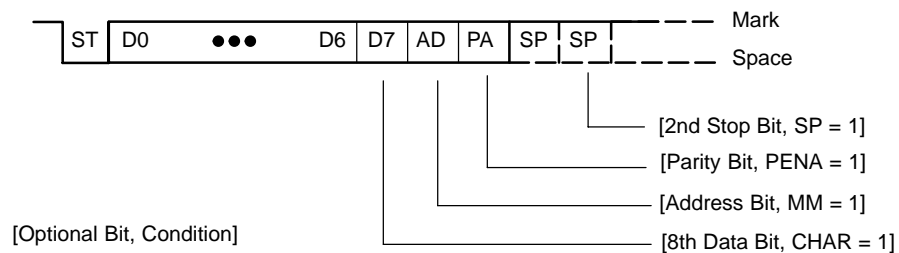
- 1) Set SWRST (`BIS.B #SWRST, &UxCTL`)
- 2) Initialize all USART registers with SWRST = 1 (including UxCTL)
- 3) Enable USART module via the MEx SFRs (URXEx and/or UTXEx)
- 4) Clear SWRST via software (`BIC.B #SWRST, &UxCTL`)
- 5) Enable interrupts (optional) via the IEx SFRs (URXIE<sub>x</sub> and/or UTXIE<sub>x</sub>)

Failure to follow this process may result in unpredictable USART behavior.

### 14.2.2 Character Format

The UART character format, shown in Figure 14–2, consists of a start bit, seven or eight data bits, an even/odd/no parity bit, an address bit (address-bit mode), and one or two stop bits. The bit period is defined by the selected clock source and setup of the baud rate registers.

Figure 14–2. Character Format



### 14.2.3 Asynchronous Communication Formats

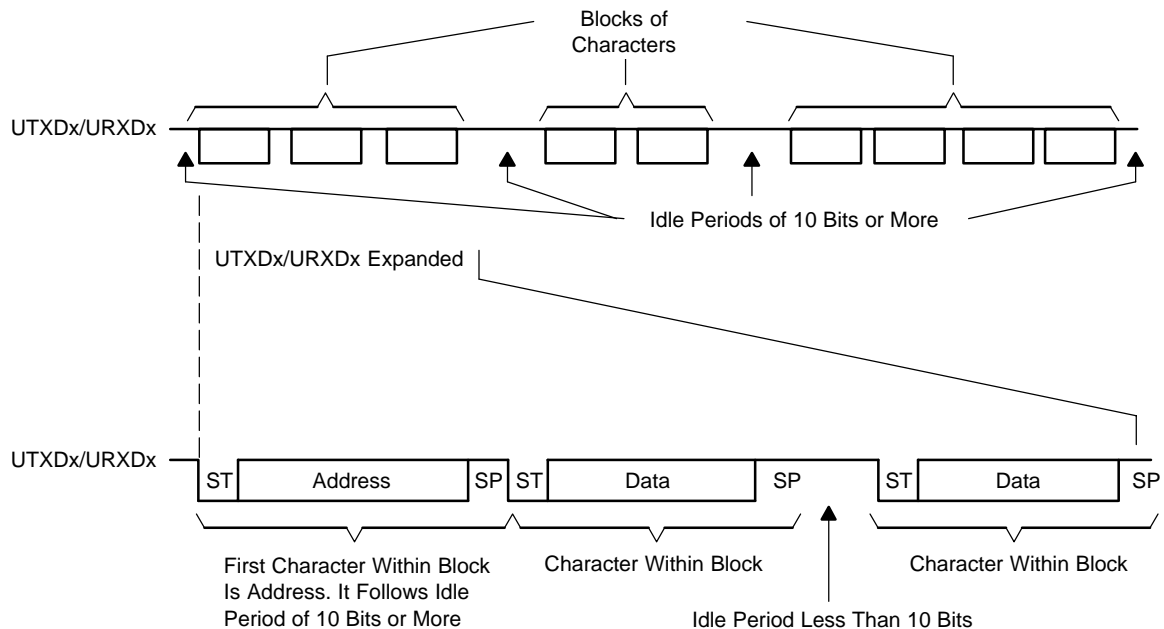
When two devices communicate asynchronously, the idle-line format is used for the protocol. When three or more devices communicate, the USART supports the idle-line and address-bit multiprocessor communication formats.

#### Idle-Line Multiprocessor Format

When MM = 0, the idle-line multiprocessor format is selected. Blocks of data are separated by an idle time on the transmit or receive lines as shown in Figure 14–3. An idle receive line is detected when 10 or more continuous ones (marks) are received after the first stop bit of a character. When two stop bits are used for the idle line the second stop bit is counted as the first mark bit of the idle period.

The first character received after an idle period is an address character. The RXWAKE bit is used as an address tag for each block of characters. In the idle-line multiprocessor format, this bit is set when a received character is an address and is transferred to UxRXBUF.

Figure 14–3. Idle-Line Format



The URXWIE bit is used to control data reception in the idle-line multiprocessor format. When the URXWIE bit is set, all non-address characters are assembled but not transferred into the UxRXBUF, and interrupts are not generated. When an address character is received, the receiver is temporarily activated to transfer the character to UxRXBUF and sets the URXIFGx interrupt flag. Any applicable error flag is also set. The user can then validate the received address.

If an address is received, user software can validate the address and must reset URXWIE to continue receiving data. If URXWIE remains set, only address characters will be received. The URXWIE bit is not modified by the USART hardware automatically.

For address transmission in idle-line multiprocessor format, a precise idle period can be generated by the USART to generate address character identifiers on UTXDx. The wake-up temporary (WUT) flag is an internal flag double-buffered with the user-accessible TXWAKE bit. When the transmitter is loaded from UxTXBUF, WUT is also loaded from TXWAKE resetting the TXWAKE bit.

The following procedure sends out an idle frame to indicate an address character will follow:

- 1) Set TXWAKE, then write any character to UxTXBUF. UxTXBUF must be ready for new data (UTXIFGx = 1).

The TXWAKE value is shifted to WUT and the contents of UxTXBUF are shifted to the transmit shift register when the shift register is ready for new data. This sets WUT, which suppresses the start, data, and parity bits of a normal transmission, then transmits an idle period of exactly 11 bits. When two stop bits are used for the idle line, the second stop bit is counted as the first mark bit of the idle period. TXWAKE is reset automatically.

- 2) Write desired address character to UxTXBUF. UxTXBUF must be ready for new data (UTXIFGx = 1).

The new character representing the specified address is shifted out following the address-identifying idle period on UTXDx. Writing the first “don’t care” character to UxTXBUF is necessary in order to shift the TXWAKE bit to WUT and generate an idle-line condition. This data is discarded and does not appear on UTXDx.

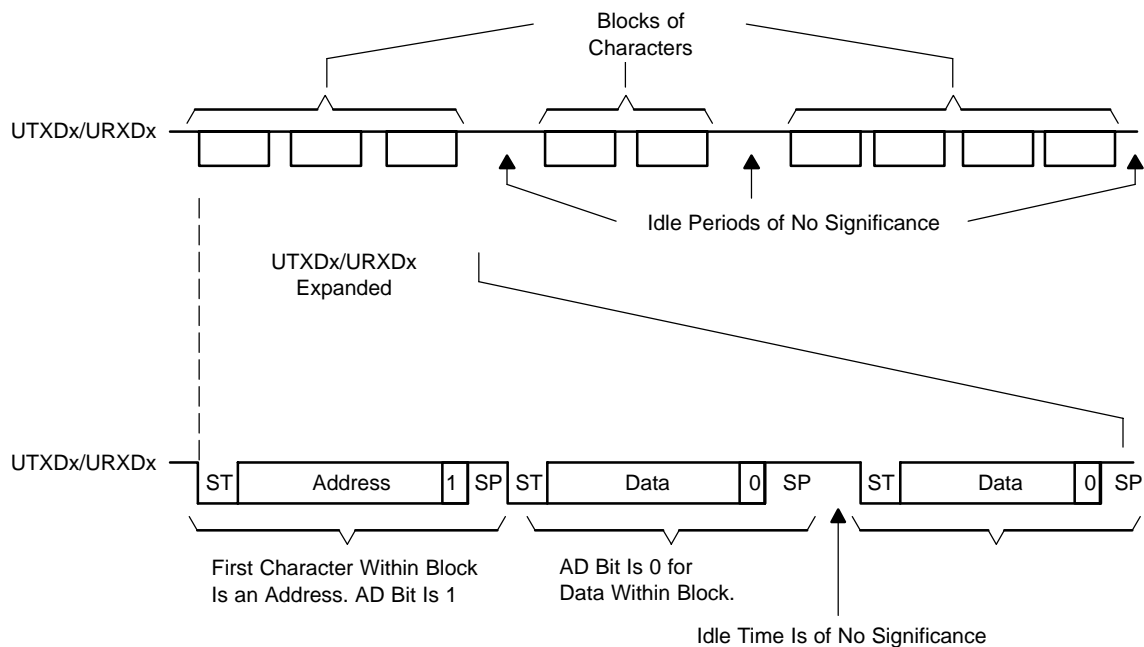
### Address-Bit Multiprocessor Format

When MM = 1, the address-bit multiprocessor format is selected. Each processed character contains an extra bit used as an address indicator shown in Figure 14–4. The first character in a block of characters carries a set address bit which indicates that the character is an address. The USART RXWAKE bit is set when a received character is a valid address character and is transferred to UxRXBUF.

The URXWIE bit is used to control data reception in the address-bit multiprocessor format. If URXWIE is set, data characters (address bit = 0) are assembled by the receiver but are not transferred to UxRXBUF and no interrupts are generated. When a character containing a set address bit is received, the receiver is temporarily activated to transfer the character to UxRXBUF and set URXIFGx. All applicable error status flags are also set.

If an address is received, user software must reset URXWIE to continue receiving data. If URXWIE remains set, only address characters (address bit = 1) will be received. The URXWIE bit is not modified by the USART hardware automatically.

Figure 14–4. Address-Bit Multiprocessor Format



For address transmission in address-bit multiprocessor mode, the address bit of a character can be controlled by writing to the TXWAKE bit. The value of the TXWAKE bit is loaded into the address bit of the character transferred from UxTXBUF to the transmit shift register, automatically clearing the TXWAKE bit. TXWAKE must not be cleared by software. It is cleared by USART hardware after it is transferred to WUT or by setting SWRST.



## Automatic Error Detection

Glitch suppression prevents the USART from being accidentally started. Any low-level on URXDx shorter than the deglitch time  $t_{\tau}$  (approximately 300 ns) will be ignored. See the device-specific datasheet for parameters.

When a low period on URXDx exceeds  $t_{\tau}$  a majority vote is taken for the start bit. If the majority vote fails to detect a valid start bit the USART halts character reception and waits for the next low period on URXDx. The majority vote is also used for each bit in a character to prevent bit errors.

The USART module automatically detects framing errors, parity errors, overrun errors, and break conditions when receiving characters. The bits FE, PE, OE, and BRK are set when their respective condition is detected. When any of these error flags are set, RXERR is also set. The error conditions are described in Table 14–1.

Table 14–1. Receive Error Conditions

Error Condition	Description
Framing error	A framing error occurs when a low stop bit is detected. When two stop bits are used, only the first stop bit is checked for framing error. When a framing error is detected, the FE bit is set.
Parity error	A parity error is a mismatch between the number of 1s in a character and the value of the parity bit. When an address bit is included in the character, it is included in the parity calculation. When a parity error is detected, the PE bit is set.
Receive overrun error	An overrun error occurs when a character is loaded into UxRXBUF before the prior character has been read. When an overrun occurs, the OE bit is set.
Break condition	A break condition is a period of 10 or more low bits received on URXDx after a missing stop bit. When a break condition is detected, the BRK bit is set. A break condition can also set the interrupt flag URXIFGx.

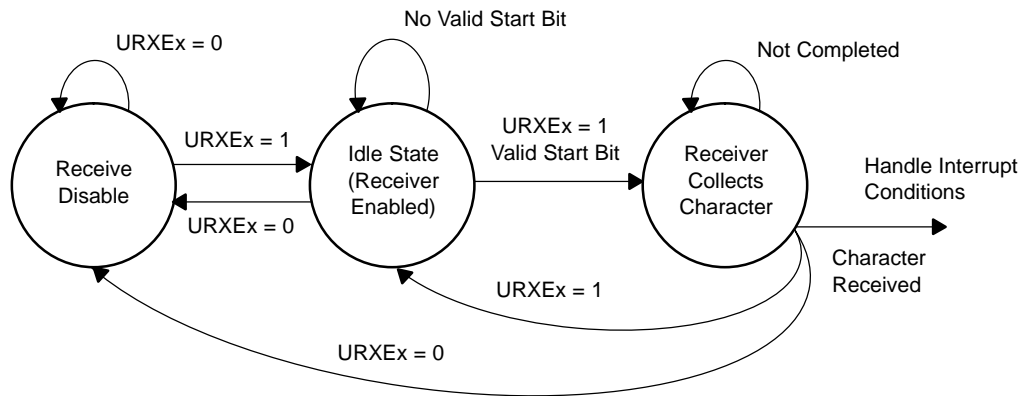
When URXEIE = 0 and a framing error, parity error, or break condition is detected, no character is received into UxRXBUF. When URXEIE = 1, characters are received into UxRXBUF and any applicable error bit is set.

When any of the FE, PE, OE, BRK, or RXERR bits is set, the bit remains set until user software resets it or UxRXBUF is read.

### 14.2.4 USART Receive Enable

The receive enable bit, URXEx, enables or disables data reception on URXDx as shown in Figure 14–5. Disabling the USART receiver stops the receive operation following completion of any character currently being received or immediately if no receive operation is active. The receive-data buffer, UxRXBUF, contains the character moved from the RX shift register after the character is received.

Figure 14–5. State Diagram of Receiver Enable



**Note: Re-Enabling the Receiver (Setting URXEx): UART Mode**

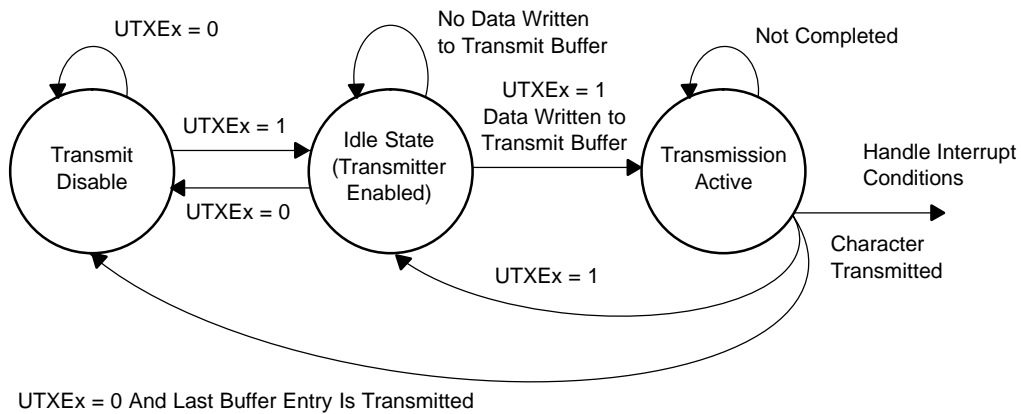
When the receiver is disabled (URXEx = 0), re-enabling the receiver (URXEx = 1) is asynchronous to any data stream that may be present on URXDx at the time. Synchronization can be performed by testing for an idle line condition before receiving a valid character (see URXWIE).

### 14.2.5 USART Transmit Enable

When UTXEx is set, the USART transmitter is enabled. Transmission is initiated by writing data to UxTXBUF. The data is then moved to the transmit shift register on the next BITCLK after the TX shift register is empty, and transmission begins. This process is shown in Figure 14–6.

When the UTXEx bit is reset the transmitter is stopped. Any data moved to UxTXBUF and any active transmission of data currently in the transmit shift register prior to clearing UTXEx will continue until all data transmission is completed.

Figure 14–6. State Diagram of Transmitter Enable



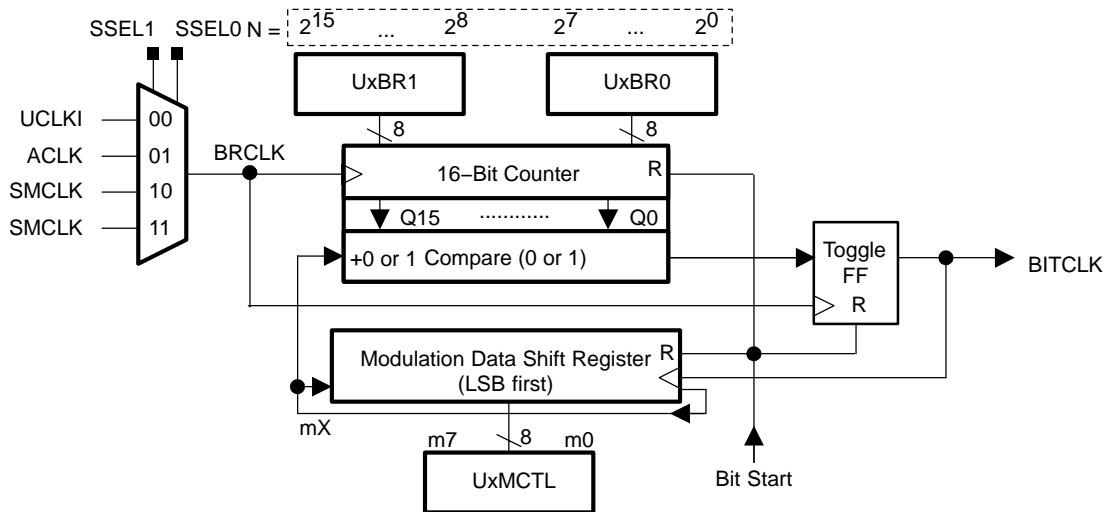
When the transmitter is enabled (UTXEx = 1), data should not be written to UxTXBUF unless it is ready for new data indicated by UTXIFGx = 1. Violation can result in an erroneous transmission if data in UxTXBUF is modified as it is being moved into the TX shift register.

It is recommended that the transmitter be disabled (UTXEx = 0) only after any active transmission is complete. This is indicated by a set transmitter empty bit (TXEPT = 1). Any data written to UxTXBUF while the transmitter is disabled will be held in the buffer but will not be moved to the transmit shift register or transmitted. Once UTXEx is set, the data in the transmit buffer is immediately loaded into the transmit shift register and character transmission resumes.

### 14.2.6 UART Baud Rate Generation

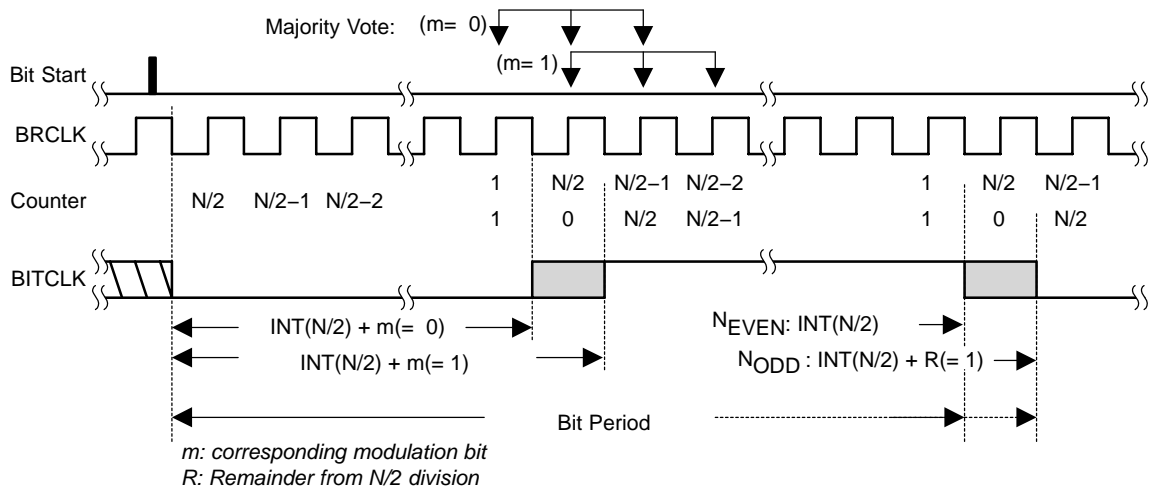
The USART baud rate generator is capable of producing standard baud rates from non-standard source frequencies. The baud rate generator uses one prescaler/divider and a modulator as shown in Figure 14–7. This combination supports fractional divisors for baud rate generation. The maximum USART baud rate is one-third the UART source clock frequency BRCLK.

Figure 14–7. MSP430 Baud Rate Generator



Timing for each bit is shown in Figure 14–8. For each bit received, a majority vote is taken to determine the bit value. These samples occur at the  $N/2-1$ ,  $N/2$ , and  $N/2+1$  BRCLK periods, where  $N$  is the number of BRCLKs per BITCLK.

Figure 14–8. BITCLK Baud Rate Timing



## Baud Rate Bit Timing

The first stage of the baud rate generator is the 16-bit counter and comparator. At the beginning of each bit transmitted or received, the counter is loaded with  $INT(N/2)$  where  $N$  is the value stored in the combination of  $UxBR0$  and  $UxBR1$ . The counter reloads  $INT(N/2)$  for each bit period half-cycle, giving a total bit period of  $N$  BRCLKs. For a given BRCLK clock source, the baud rate used determines the required division factor  $N$ :

$$N = \frac{BRCLK}{\text{baud rate}}$$

The division factor  $N$  is often a non-integer value of which the integer portion can be realized by the prescaler/divider. The second stage of the baud rate generator, the modulator, is used to meet the fractional part as closely as possible. The factor  $N$  is then defined as:

$$N = UxBR + \frac{1}{n} \sum_{i=0}^{n-1} m_i$$

Where:

- $N$ : Target division factor
- $UxBR$ : 16-bit representation of registers  $UxBR0$  and  $UxBR1$
- $i$ : Bit position in the character
- $n$ : Total number of bits in the character
- $m_i$ : Data of each corresponding modulation bit (1 or 0)

$$\text{Baud rate} = \frac{BRCLK}{N} = \frac{BRCLK}{UxBR + \frac{1}{n} \sum_{i=0}^{n-1} m_i}$$

The BITCLK can be adjusted from bit to bit with the modulator to meet timing requirements when a non-integer divisor is needed. Timing of each bit is expanded by one BRCLK clock cycle if the modulator bit  $m_i$  is set. Each time a bit is received or transmitted, the next bit in the modulation control register determines the timing for that bit. A set modulation bit increases the division factor by one while a cleared modulation bit maintains the division factor given by  $UxBR$ .

The timing for the start bit is determined by  $UxBR$  plus  $m_0$ , the next bit is determined by  $UxBR$  plus  $m_1$ , and so on. The modulation sequence begins with the LSB. When the character is greater than 8 bits, the modulation sequence restarts with  $m_0$  and continues until all bits are processed.

## Determining the Modulation Value

Determining the modulation value is an interactive process. Using the timing error formula provided, beginning with the start bit, the individual bit errors are calculated with the corresponding modulator bit set and cleared. The modulation bit setting with the lower error is selected and the next bit error is calculated. This process is continued until all bit errors are minimized. When a character contains more than 8 bits, the modulation bits repeat. For example, the 9th bit of a character uses modulation bit 0.

## Transmit Bit Timing

The timing for each character is the sum of the individual bit timings. By modulating each bit, the cumulative bit error is reduced. The individual bit error can be calculated by:

$$\text{Error [\%]} = \left\{ \frac{\text{baud rate}}{\text{BRCLK}} \times \left[ (j + 1) \times \text{UxBR} + \sum_{i=0}^j m_i \right] - (j + 1) \right\} \times 100\%$$

With:

*baud rate*: Desired baud rate

*BRCLK*: Input frequency – UCLKI, ACLK, or SMCLK

*j*: Bit position - 0 for the start bit, 1 for data bit D0, and so on

*UxBR*: Division factor in registers UxBR1 and UxBR0

For example, the transmit errors for the following conditions are calculated:

Baud rate = 2400  
 BRCLK = 32,768 Hz (ACLK)  
 UxBR = 13, since the ideal division factor is 13.65  
 UxMCTL = 6Bh: m7=0, m6=1, m5=1, m4=0, m3=1, m2=0, m1=1, and m0=1. The LSB of UxMCTL is used first.

$$\text{Start bit Error [\%]} = \left( \frac{\text{baud rate}}{\text{BRCLK}} \times ((0 + 1) \times \text{UxBR} + 1) - 1 \right) \times 100\% = 2.54\%$$

$$\text{Data bit D0 Error [\%]} = \left( \frac{\text{baud rate}}{\text{BRCLK}} \times ((1 + 1) \times \text{UxBR} + 2) - 2 \right) \times 100\% = 5.08\%$$

$$\text{Data bit D1 Error [\%]} = \left( \frac{\text{baud rate}}{\text{BRCLK}} \times ((2 + 1) \times \text{UxBR} + 2) - 3 \right) \times 100\% = 0.29\%$$

$$\text{Data bit D2 Error [\%]} = \left( \frac{\text{baud rate}}{\text{BRCLK}} \times ((3 + 1) \times \text{UxBR} + 3) - 4 \right) \times 100\% = 2.83\%$$

$$\text{Data bit D3 Error [\%]} = \left( \frac{\text{baud rate}}{\text{BRCLK}} \times ((4 + 1) \times \text{UxBR} + 3) - 5 \right) \times 100\% = -1.95\%$$

$$\text{Data bit D4 Error [\%]} = \left( \frac{\text{baud rate}}{\text{BRCLK}} \times ((5 + 1) \times \text{UxBR} + 4) - 6 \right) \times 100\% = 0.59\%$$

$$\text{Data bit D5 Error [\%]} = \left( \frac{\text{baud rate}}{\text{BRCLK}} \times ((6 + 1) \times \text{UxBR} + 5) - 7 \right) \times 100\% = 3.13\%$$

$$\text{Data bit D6 Error [\%]} = \left( \frac{\text{baud rate}}{\text{BRCLK}} \times ((7 + 1) \times \text{UxBR} + 5) - 8 \right) \times 100\% = -1.66\%$$

$$\text{Data bit D7 Error [\%]} = \left( \frac{\text{baud rate}}{\text{BRCLK}} \times ((8 + 1) \times \text{UxBR} + 6) - 9 \right) \times 100\% = 0.88\%$$

$$\text{Parity bit Error [\%]} = \left( \frac{\text{baud rate}}{\text{BRCLK}} \times ((9 + 1) \times \text{UxBR} + 7) - 10 \right) \times 100\% = 3.42\%$$

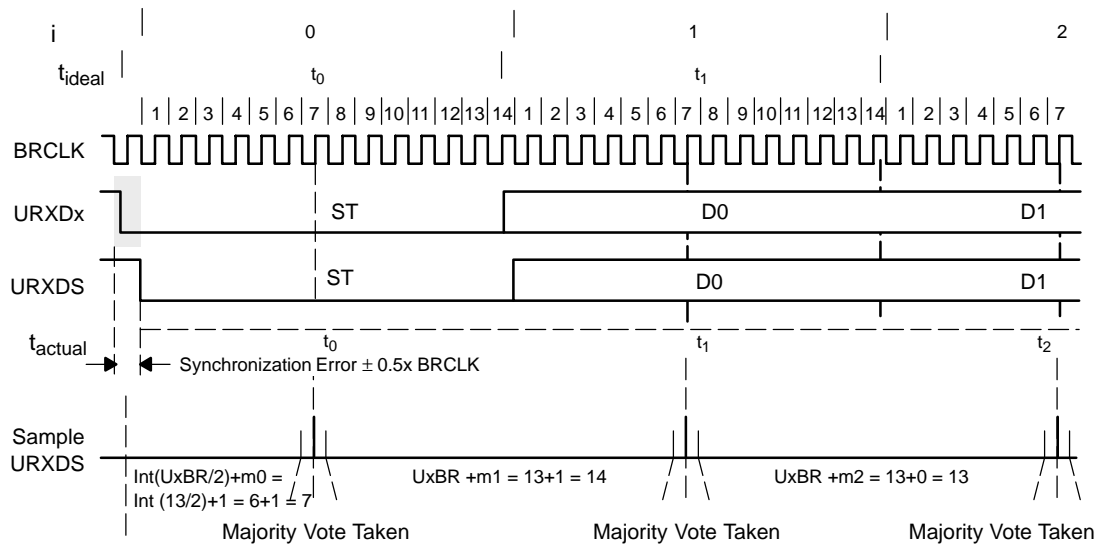
$$\text{Stop bit 1 Error [\%]} = \left( \frac{\text{baud rate}}{\text{BRCLK}} \times ((10 + 1) \times \text{UxBR} + 7) - 11 \right) \times 100\% = -1.37\%$$

The results show the maximum per-bit error to be 5.08% of a BITCLK period.

## Receive Bit Timing

Receive timing consists of two error sources. The first is the bit-to-bit timing error. The second is the error between a start edge occurring and the start edge being accepted by the USART. Figure 14–9 shows the asynchronous timing errors between data on the URXDx pin and the internal baud-rate clock.

Figure 14–9. Receive Error



The ideal start bit timing  $t_{ideal(0)}$  is half the baud-rate timing  $t_{baud\ rate}$  because the bit is tested in the middle of its period. The ideal baud rate timing  $t_{ideal(i)}$  for the remaining character bits is the baud rate timing  $t_{baud\ rate}$ . The individual bit errors can be calculated by:

$$Error\ [\%] = \left[ \frac{baud\ rate}{BRCLK} \times \left\{ 2 \times \left[ m_0 + int\left(\frac{UxBR}{2}\right) \right] + \left( i \times UxBR + \sum_{i=1}^j m_i \right) \right\} - 1 - j \right] \times 100\%$$

Where:

*baud rate* is the required baud rate

*BRCLK* is the input frequency—selected for UCLK, ACLK, or SMCLK

*j* = 0 for the start bit, 1 for data bit D0, and so on

*UxBR* is the division factor in registers UxBR1 and UxBR0

For example, the receive errors for the following conditions are calculated:

Baud rate = 2400  
 BRCLK = 32,768 Hz (ACLK)  
 UxBR = 13, since the ideal division factor is 13.65  
 UxMCTL = 6B:m7=0, m6=1, m5=1, m4=0, m3=1, m2=0, m1=1 and m0=1 The LSB of UxMCTL is used first.

$$\text{Start bit Error [\%]} = \left( \frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (0 \times \text{UxBR} + 0)] - 1 - 0 \right) \times 100\% = 2.54\%$$

$$\text{Data bit D0 Error [\%]} = \left( \frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (1 \times \text{UxBR} + 1)] - 1 - 1 \right) \times 100\% = 5.08\%$$

$$\text{Data bit D1 Error [\%]} = \left( \frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (2 \times \text{UxBR} + 1)] - 1 - 2 \right) \times 100\% = 0.29\%$$

$$\text{Data bit D2 Error [\%]} = \left( \frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (3 \times \text{UxBR} + 2)] - 1 - 3 \right) \times 100\% = 2.83\%$$

$$\text{Data bit D3 Error [\%]} = \left( \frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (4 \times \text{UxBR} + 2)] - 1 - 4 \right) \times 100\% = -1.95\%$$

$$\text{Data bit D4 Error [\%]} = \left( \frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (5 \times \text{UxBR} + 3)] - 1 - 5 \right) \times 100\% = 0.59\%$$

$$\text{Data bit D5 Error [\%]} = \left( \frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (6 \times \text{UxBR} + 4)] - 1 - 6 \right) \times 100\% = 3.13\%$$

$$\text{Data bit D6 Error [\%]} = \left( \frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (7 \times \text{UxBR} + 4)] - 1 - 7 \right) \times 100\% = -1.66\%$$

$$\text{Data bit D7 Error [\%]} = \left( \frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (8 \times \text{UxBR} + 5)] - 1 - 8 \right) \times 100\% = 0.88\%$$

$$\text{Parity bit Error [\%]} = \left( \frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (9 \times \text{UxBR} + 6)] - 1 - 9 \right) \times 100\% = 3.42\%$$

$$\text{Stop bit 1 Error [\%]} = \left( \frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (10 \times \text{UxBR} + 6)] - 1 - 10 \right) \times 100\% = -1.37\%$$

The results show the maximum per-bit error to be 5.08% of a BITCLK period.



## Typical Baud Rates and Errors

Standard baud rate frequency data for UxBRx and UxMCTL are listed in Table 14–2 for a 32,768-Hz watch crystal (ACLK) and a typical 1,048,576-Hz SMCLK.

The receive error is the accumulated time versus the ideal scanning time in the middle of each bit. The transmit error is the accumulated timing error versus the ideal time of the bit period.

Table 14–2. Commonly Used Baud Rates, Baud Rate Data, and Errors

Baud Rate	Divide by		A: BRCLK = 32,768 Hz						B: BRCLK = 1,048,576 Hz				
	A:	B:	UxBR1	UxBR0	UxMCTL	Max. TX Error %	Max. RX Error %	Synchr. RX Error %	UxBR1	UxBR0	UxMCTL	Max. TX Error %	Max. RX Error %
1200	27.31	873.81	0	1B	03	-4/3	-4/3	±2	03	69	FF	0/0.3	±2
2400	13.65	436.91	0	0D	6B	-6/3	-6/3	±4	01	B4	FF	0/0.3	±2
4800	6.83	218.45	0	06	6F	-9/11	-9/11	±7	0	DA	55	0/0.4	±2
9600	3.41	109.23	0	03	4A	-21/12	-21/12	±15	0	6D	03	-0.4/1	±2
19,200		54.61							0	36	6B	-0.2/2	±2
38,400		27.31							0	1B	03	-4/3	±2
76,800		13.65							0	0D	6B	-6/3	±4
115,200		9.1							0	09	08	-5/7	±7

### 14.2.7 USART Interrupts

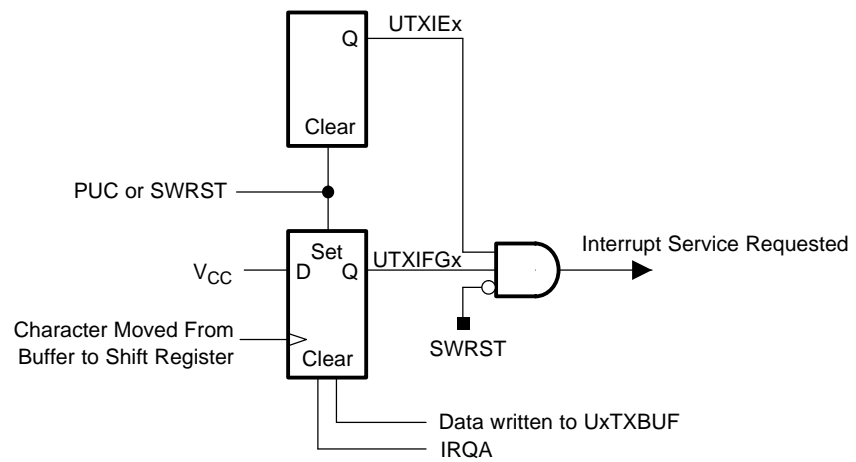
The USART has one interrupt vector for transmission and one interrupt vector for reception.

#### USART Transmit Interrupt Operation

The UTXIFGx interrupt flag is set by the transmitter to indicate that UxTXBUF is ready to accept another character. An interrupt request is generated if UTXIEx and GIE are also set. UTXIFGx is automatically reset if the interrupt request is serviced or if a character is written to UxTXBUF.

UTXIFGx is set after a PUC or when SWRST = 1. UTXIEx is reset after a PUC or when SWRST = 1. The operation is shown in Figure 14–10.

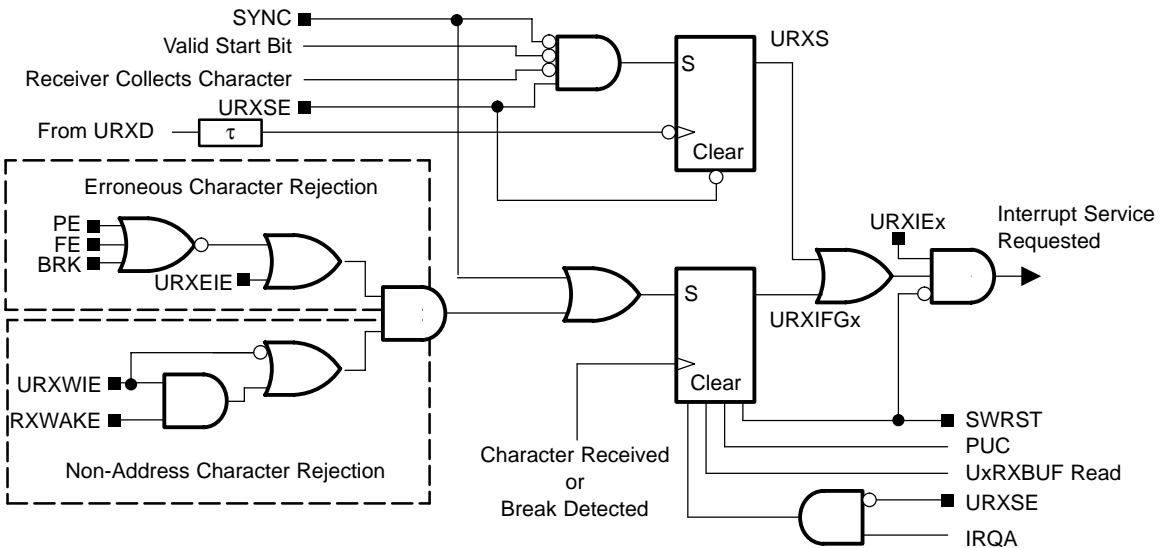
Figure 14–10. Transmit Interrupt Operation



## USART Receive Interrupt Operation

The URXIFGx interrupt flag is set each time a character is received and loaded into UxRXBUF. An interrupt request is generated if URXIE<sub>x</sub> and GIE are also set. URXIFG<sub>x</sub> and URXIE<sub>x</sub> are reset by a system reset PUC signal or when SWRST = 1. URXIFG<sub>x</sub> is automatically reset if the pending interrupt is served (when URXSE = 0) or when UxRXBUF is read. The operation is shown in Figure 14–11.

Figure 14–11. Receive Interrupt Operation



URXEIE is used to enable or disable erroneous characters from setting URXIFG<sub>x</sub>. When using multiprocessor addressing modes, URXWIE is used to auto-detect valid address characters and reject unwanted data characters.

Two types of characters do not set URXIFG<sub>x</sub>:

- Erroneous characters when URXEIE = 0
- Non-address characters when URXWIE = 1

When URXEIE = 1 a break condition will set the BRK bit and the URXIFG<sub>x</sub> flag.

## Receive-Start Edge Detect Operation

The URXSE bit enables the receive start-edge detection feature. The recommended usage of the receive-start edge feature is when BRCLK is sourced by the DCO and when the DCO is off because of low-power mode operation. The ultra-fast turn-on of the DCO allows character reception after the start edge detection.

When URXSE, URXIE<sub>x</sub> and GIE are set and a start edge occurs on URXD<sub>x</sub>, the internal signal URXS will be set. When URXS is set, a receive interrupt request is generated but URXIFG<sub>x</sub> is not set. User software in the receive interrupt service routine can test URXIFG<sub>x</sub> to determine the source of the interrupt. When URXIFG<sub>x</sub> = 0 a start edge was detected and when URXIFG<sub>x</sub> = 1 a valid character (or break) was received.

When the ISR determines the interrupt request was from a start edge, user software toggles URXSE, and must enable the BRCLK source by returning from the ISR to active mode or to a low-power mode where the source is active. If the ISR returns to a low-power mode where the BRCLK source is inactive, the character will not be received. Toggling URXSE clears the URXS signal and re-enables the start edge detect feature for future characters. See chapter *System Resets, Interrupts, and Operating Modes* for information on entering and exiting low-power modes.

The now active BRCLK allows the USART to receive the balance of the character. After the full character is received and moved to UxRXBUF, URXIFG<sub>x</sub> is set and an interrupt service is again requested. Upon ISR entry, URXIFG<sub>x</sub> = 1 indicating a character was received. The URXIFG<sub>x</sub> flag is cleared when user software reads UxRXBUF.

```

; Interrupt handler for start condition and
; Character receive. BRCLK = DCO.

U0RX_Int BIT.B #URXIFG0,&IFG2 ; Test URXIFGx to determine
      JNE ST_COND ; If start or character
      MOV.B &UxRXBUF,dst ; Read buffer
      ... ;
      RETI ;

ST_COND BIC.B #URXSE,&U0TCTL ; Clear URXS signal
      BIS.B #URXSE,&U0TCTL ; Re-enable edge detect
      BIC #SCG0+SCG1,0(SP) ; Enable BRCLK = DCO
      RETI ;

```

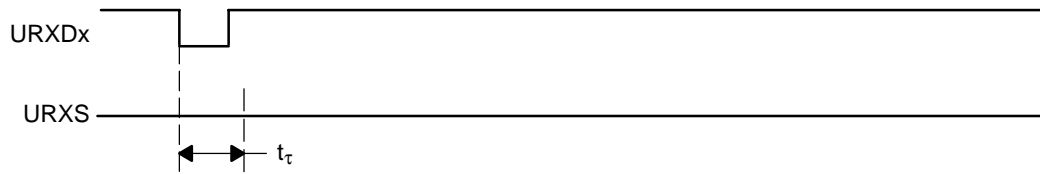
### **Note: Break Detect With Halted UART Clock**

When using the receive start-edge detect feature a break condition cannot be detected when the BRCLK source is off.

**Receive-Start Edge Detect Conditions**

When URXSE = 1, glitch suppression prevents the USART from being accidentally started. Any low-level on URXDx shorter than the deglitch time  $t_{\tau}$  (approximately 300 ns) will be ignored by the USART and no interrupt request will be generated as shown in Figure 14–12. See the device-specific datasheet for parameters.

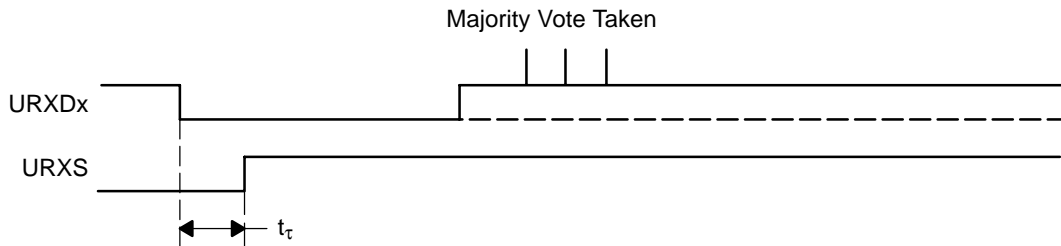
Figure 14–12. Glitch Suppression, USART Receive Not Started



When a glitch is longer than  $t_{\tau}$ , or a valid start bit occurs on URXDx, the USART receive operation is started and a majority vote is taken as shown in Figure 14–13. If the majority vote fails to detect a start bit the USART halts character reception.

If character reception is halted, an active BRCLK is not necessary. A time-out period longer than the character receive duration can be used by software to indicate that a character was not received in the expected time and the software can disable BRCLK.

Figure 14–13. Glitch Suppression, USART Activated



### 14.3 USART Registers: UART Mode

Table 14–3 lists the registers for all devices implementing a USART module. Table 14–4 applies only to devices with a second USART module, USART1.

Table 14–3. USART0 Control and Status Registers

Register	Short Form	Register Type	Address	Initial State
USART control register	U0CTL	Read/write	070h	001h with PUC
Transmit control register	U0TCTL	Read/write	071h	001h with PUC
Receive control register	U0RCTL	Read/write	072h	000h with PUC
Modulation control register	U0MCTL	Read/write	073h	Unchanged
Baud rate control register 0	U0BR0	Read/write	074h	Unchanged
Baud rate control register 1	U0BR1	Read/write	075h	Unchanged
Receive buffer register	U0RXBUF	Read	076h	Unchanged
Transmit buffer register	U0TXBUF	Read/write	077h	Unchanged
SFR module enable register 1	ME1	Read/write	004h	000h with PUC
SFR interrupt enable register 1	IE1	Read/write	000h	000h with PUC
SFR interrupt flag register 1	IFG1	Read/write	002h	082h with PUC

Table 14–4. USART1 Control and Status Registers

Register	Short Form	Register Type	Address	Initial State
USART control register	U1CTL	Read/write	078h	001h with PUC
Transmit control register	U1TCTL	Read/write	079h	001h with PUC
Receive control register	U1RCTL	Read/write	07Ah	000h with PUC
Modulation control register	U1MCTL	Read/write	07Bh	Unchanged
Baud rate control register 0	U1BR0	Read/write	07Ch	Unchanged
Baud rate control register 1	U1BR1	Read/write	07Dh	Unchanged
Receive buffer register	U1RXBUF	Read	07Eh	Unchanged
Transmit buffer register	U1TXBUF	Read/write	07Fh	Unchanged
SFR module enable register 2	ME2	Read/write	005h	000h with PUC
SFR interrupt enable register 2	IE2	Read/write	001h	000h with PUC
SFR interrupt flag register 2	IFG2	Read/write	003h	020h with PUC

**Note: Modifying SFR bits**

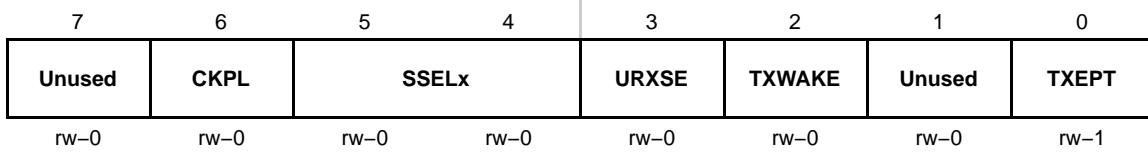
To avoid modifying control bits of other modules, it is recommended to set or clear the IEx and IFGx bits using `BIS . B` or `BIC . B` instructions, rather than `MOV . B` or `CLR . B` instructions.

## UxCTL, USART Control Register

7	6	5	4	3	2	1	0
<b>PENA</b>	<b>PEV</b>	<b>SPB</b>	<b>CHAR</b>	<b>LISTEN</b>	<b>SYNC</b>	<b>MM</b>	<b>SWRST</b>
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1

<b>PENA</b>	Bit 7	Parity enable 0 Parity disabled. 1 Parity enabled. Parity bit is generated (UTXDx) and expected (URXDx). In address-bit multiprocessor mode, the address bit is included in the parity calculation.
<b>PEV</b>	Bit 6	Parity select. PEV is not used when parity is disabled. 0 Odd parity 1 Even parity
<b>SPB</b>	Bit 5	Stop bit select. Number of stop bits transmitted. The receiver always checks for one stop bit. 0 One stop bit 1 Two stop bits
<b>CHAR</b>	Bit 4	Character length. Selects 7-bit or 8-bit character length. 0 7-bit data 1 8-bit data
<b>LISTEN</b>	Bit 3	Listen enable. The LISTEN bit selects loopback mode. 0 Disabled 1 Enabled. UTXDx is internally fed back to the receiver.
<b>SYNC</b>	Bit 2	Synchronous mode enable 0 UART mode 1 SPI Mode
<b>MM</b>	Bit 1	Multiprocessor mode select 0 Idle-line multiprocessor protocol 1 Address-bit multiprocessor protocol
<b>SWRST</b>	Bit 0	Software reset enable 0 Disabled. USART reset released for operation 1 Enabled. USART logic held in reset state

**UxTCTL, USART Transmit Control Register**



<b>Unused</b>	Bit 7	Unused
<b>CKPL</b>	Bit 6	Clock polarity select 0 UCLKI = UCLK 1 UCLKI = inverted UCLK
<b>SSELx</b>	Bits 5-4	Source select. These bits select the BRCLK source clock. 00 UCLKI 01 ACLK 10 SMCLK 11 SMCLK
<b>URXSE</b>	Bit 3	UART receive start-edge. The bit enables the UART receive start-edge feature. 0 Disabled 1 Enabled
<b>TXWAKE</b>	Bit 2	Transmitter wake 0 Next frame transmitted is data 1 Next frame transmitted is an address
<b>Unused</b>	Bit 1	Unused
<b>TXEPT</b>	Bit 0	Transmitter empty flag 0 UART is transmitting data and/or data is waiting in UxTXBUF 1 Transmitter shift register and UxTXBUF are empty or SWRST=1

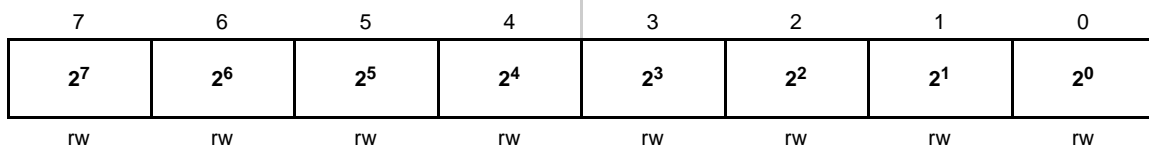


## UxRCTL, USART Receive Control Register

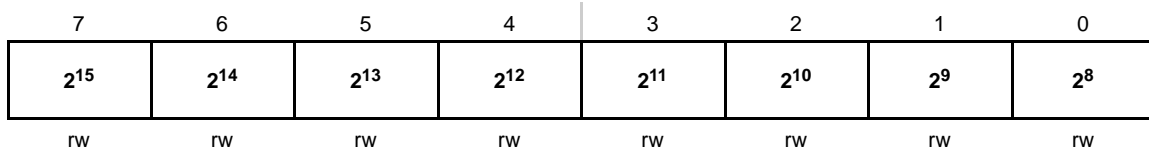
7	6	5	4	3	2	1	0
<b>FE</b>	<b>PE</b>	<b>OE</b>	<b>BRK</b>	<b>URXEIE</b>	<b>URXWIE</b>	<b>RXWAKE</b>	<b>RXERR</b>
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

<b>FE</b>	Bit 7	Framing error flag 0 No error 1 Character received with low stop bit
<b>PE</b>	Bit 6	Parity error flag. When PENA = 0, PE is read as 0. 0 No error 1 Character received with parity error
<b>OE</b>	Bit 5	Overrun error flag. This bit is set when a character is transferred into UxRXBUF before the previous character was read. 0 No error 1 Overrun error occurred
<b>BRK</b>	Bit 4	Break detect flag 0 No break condition 1 Break condition occurred
<b>URXEIE</b>	Bit 3	Receive erroneous-character interrupt-enable 0 Erroneous characters rejected and URXIFGx is not set 1 Erroneous characters received will set URXIFGx
<b>URXWIE</b>	Bit 2	Receive wake-up interrupt-enable. This bit enables URXIFGx to be set when an address character is received. When URXEIE = 0, an address character will not set URXIFGx if it is received with errors. 0 All received characters set URXIFGx 1 Only received address characters set URXIFGx
<b>RXWAKE</b>	Bit 1	Receive wake-up flag 0 Received character is data 1 Received character is an address
<b>RXERR</b>	Bit 0	Receive error flag. This bit indicates a character was received with error(s). When RXERR = 1, on or more error flags (FE,PE,OE, BRK) is also set. RXERR is cleared when UxRXBUF is read. 0 No receive errors detected 1 Receive error detected

**UxBR0, USART Baud Rate Control Register 0**

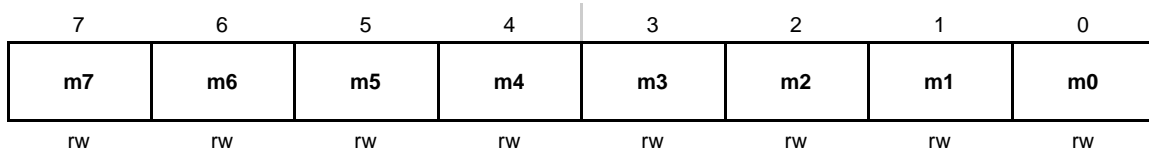


**UxBR1, USART Baud Rate Control Register 1**



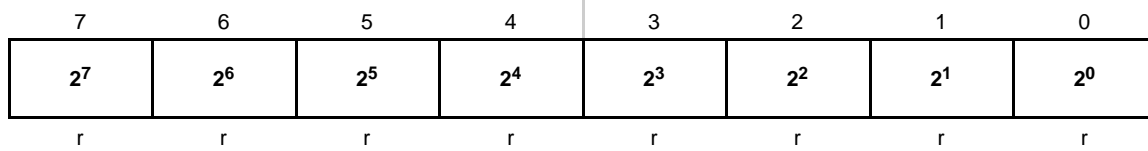
**UxBRx**                      The valid baud-rate control range is  $3 \leq UxBR < 0FFFFh$ , where  $UxBR = \{UxBR1+UxBR0\}$ . Unpredictable receive and transmit timing occurs if  $UxBR < 3$ .

**UxMCTL, USART Modulation Control Register**



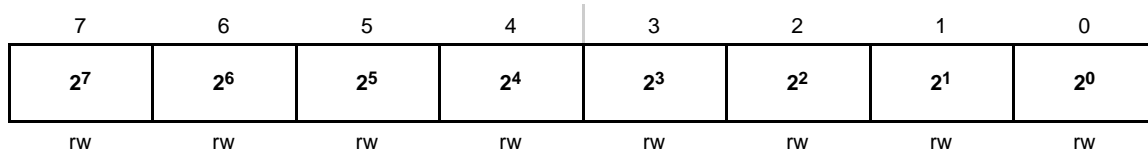
**UxMCTLx**      Bits              Modulation bits. These bits select the modulation for BRCLK.  
                          7-0

### UxRXBUF, USART Receive Buffer Register



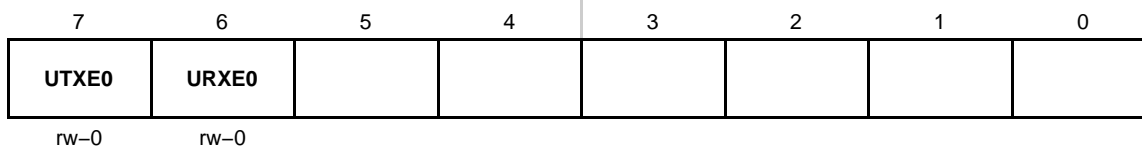
**UxRXBUFx** Bits 7–0 The receive-data buffer is user accessible and contains the last received character from the receive shift register. Reading UxRXBUF resets the receive-error bits, the RXWAKE bit, and URXIFGx. In 7-bit data mode, UxRXBUF is LSB justified and the MSB is always reset.

### UxTXBUF, USART Transmit Buffer Register



**UxTXBUFx** Bits 7–0 The transmit data buffer is user accessible and holds the data waiting to be moved into the transmit shift register and transmitted on UTXDx. Writing to the transmit data buffer clears UTXIFGx. The MSB of UxTXBUF is not used for 7-bit data and is reset.

### ME1, Module Enable Register 1

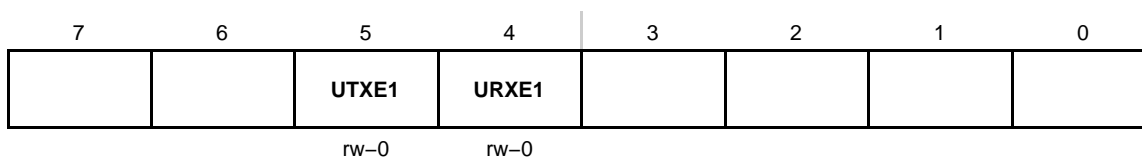


**UTXE0**    Bit 7    USART0 transmit enable. This bit enables the transmitter for USART0.  
 0    Module not enabled  
 1    Module enabled

**URXE0**    Bit 6    USART0 receive enable. This bit enables the receiver for USART0.  
 0    Module not enabled  
 1    Module enabled

Bits    These bits may be used by other modules. See device-specific datasheet.  
 5-0

### ME2, Module Enable Register 2



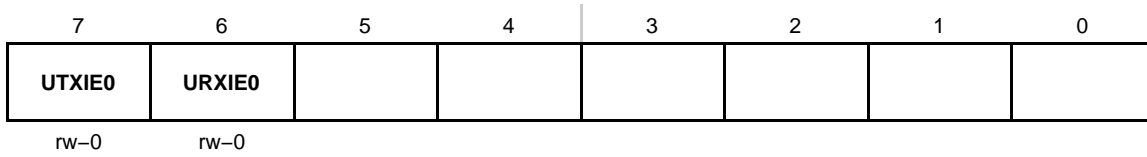
Bits    These bits may be used by other modules. See device-specific datasheet.  
 7-6

**UTXE1**    Bit 5    USART1 transmit enable. This bit enables the transmitter for USART1.  
 0    Module not enabled  
 1    Module enabled

**URXE1**    Bit 4    USART1 receive enable. This bit enables the receiver for USART1.  
 0    Module not enabled  
 1    Module enabled

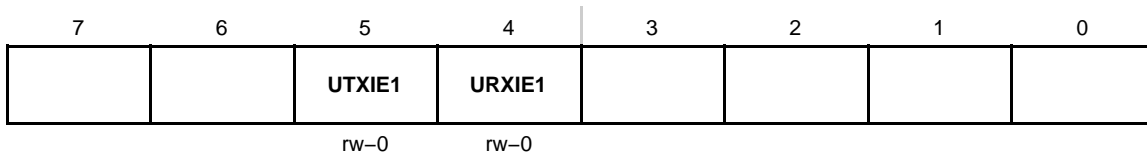
Bits    These bits may be used by other modules. See device-specific datasheet.  
 3-0

### IE1, Interrupt Enable Register 1



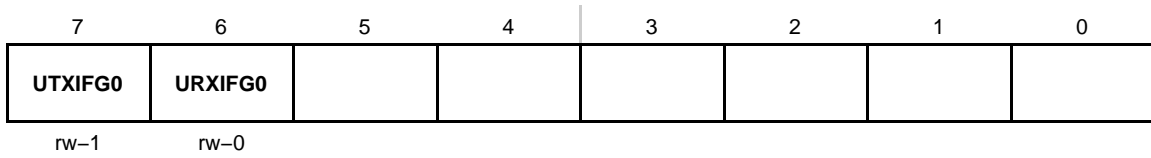
- UTXIE0** Bit 7 USART0 transmit interrupt enable. This bit enables the UTXIFG0 interrupt.  
 0 Interrupt not enabled  
 1 Interrupt enabled
- URXIE0** Bit 6 USART0 receive interrupt enable. This bit enables the URXIFG0 interrupt.  
 0 Interrupt not enabled  
 1 Interrupt enabled
- Bits 5-0 These bits may be used by other modules. See device-specific datasheet.

### IE2, Interrupt Enable Register 2



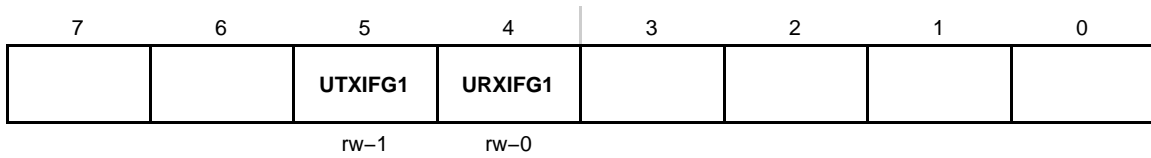
- Bits 7-6 These bits may be used by other modules. See device-specific datasheet.
- UTXIE1** Bit 5 USART1 transmit interrupt enable. This bit enables the UTXIFG1 interrupt.  
 0 Interrupt not enabled  
 1 Interrupt enabled
- URXIE1** Bit 4 USART1 receive interrupt enable. This bit enables the URXIFG1 interrupt.  
 0 Interrupt not enabled  
 1 Interrupt enabled
- Bits 3-0 These bits may be used by other modules. See device-specific datasheet.

### IFG1, Interrupt Flag Register 1



- UTXIFG0†** Bit 7 USART0 transmit interrupt flag. UTXIFG0 is set when U0TXBUF is empty.  
 0 No interrupt pending  
 1 Interrupt pending
- URXIFG0†** Bit 6 USART0 receive interrupt flag. URXIFG0 is set when U0RXBUF has received a complete character.  
 0 No interrupt pending  
 1 Interrupt pending
- Bits 5-0 These bits may be used by other modules. See device-specific datasheet.

### IFG2, Interrupt Flag Register 2



- Bits 7-6 These bits may be used by other modules. See device-specific datasheet.
- UTXIFG1** Bit 5 USART1 transmit interrupt flag. UTXIFG1 is set when U1TXBUF empty.  
 0 No interrupt pending  
 1 Interrupt pending
- URXIFG1** Bit 4 USART1 receive interrupt flag. URXIFG1 is set when U1RXBUF has received a complete character.  
 0 No interrupt pending  
 1 Interrupt pending
- Bits 3-0 These bits may be used by other modules. See device-specific datasheet.

# USART Peripheral Interface, SPI Mode

---

---

---

The universal synchronous/asynchronous receive/transmit (USART) peripheral interface supports two serial modes with one hardware module. This chapter discusses the operation of the synchronous peripheral interface or SPI mode. USART0 is implemented on the MSP430x42x and MSP430x43x devices. In addition to USART0, the MSP430x44x devices implement a second identical USART module, USART1.

<b>Topic</b>	<b>Page</b>
<b>15.1 USART Introduction: SPI Mode</b> .....	<b>15-2</b>
<b>15.2 USART Operation: SPI Mode</b> .....	<b>15-4</b>
<b>15.3 USART Registers: SPI Mode</b> .....	<b>15-13</b>

## 15.1 USART Introduction: SPI Mode

In synchronous mode, the USART connects the MSP430 to an external system via three or four pins: SIMO, SOMI, UCLK, and STE. SPI mode is selected when the SYNC bit is set and the I2C bit is cleared.

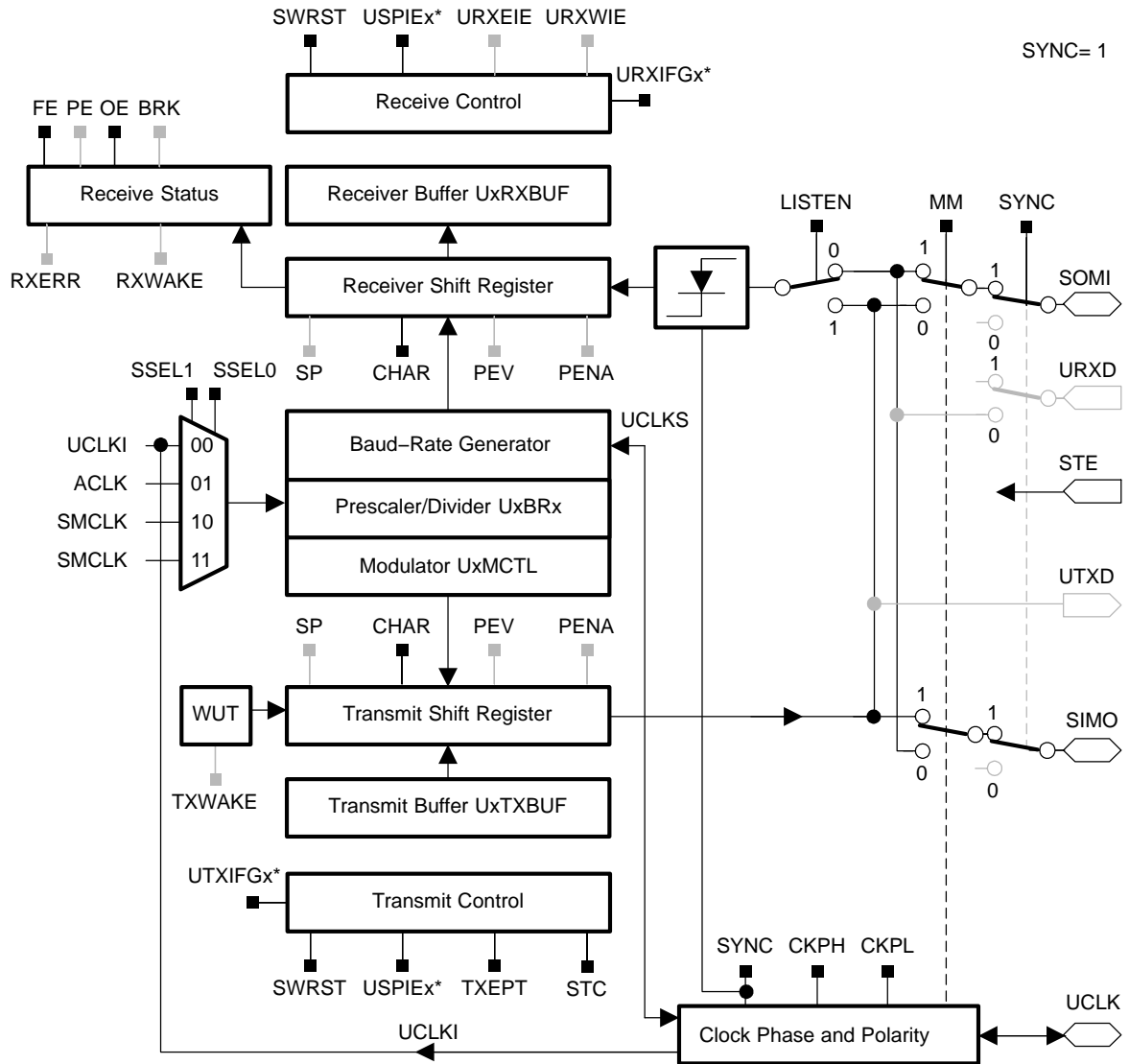
SPI mode features include:

- 7- or 8-bit data length
- 3-pin and 4-pin SPI operation
- Master or slave modes
- Independent transmit and receive shift registers
- Separate transmit and receive buffer registers
- Selectable UCLK polarity and phase control
- Programmable UCLK frequency in master mode
- Independent interrupt capability for receive and transmit

Figure 15–1 shows the USART when configured for SPI mode.



Figure 15–1. USART Block Diagram: SPI Mode



\* Refer to the device-specific datasheet for SFR locations

## 15.2 USART Operation: SPI Mode

In SPI mode, serial data is transmitted and received by multiple devices using a shared clock provided by the master. An additional pin, STE, is provided as to enable a device to receive and transmit data and is controlled by the master.

Three or four signals are used for SPI data exchange:

- SIMO** Slave in, master out  
Master mode: SIMO is the data output line.  
Slave mode: SIMO is the data input line.
- SOMI** Slave out, master in  
Master mode: SOMI is the data input line.  
Slave mode: SOMI is the data output line.
- UCLK** USART SPI clock  
Master mode: UCLK is an output.  
Slave mode: UCLK is an input.
- STE** Slave transmit enable. Used in 4-pin mode to allow multiple masters on a single bus. Not used in 3-pin mode.  
4-Pin master mode:  
When STE is high, SIMO and UCLK operate normally.  
When STE is low, SIMO and UCLK are set to the input direction.  
4-pin slave mode:  
When STE is high, RX/TX operation of the slave is disabled and SOMI is forced to the input direction.  
When STE is low, RX/TX operation of the slave is enabled and SOMI operates normally.

### 15.2.1 USART Initialization and Reset

The USART is reset by a PUC or by the SWRST bit. After a PUC, the SWRST bit is automatically set, keeping the USART in a reset condition. When set, the SWRST bit resets the URXIE<sub>x</sub>, UTXIE<sub>x</sub>, URXIFG<sub>x</sub>, OE, and FE bits and sets the UTXIFG<sub>x</sub> flag. The USPIE<sub>x</sub> bit is not altered by SWRST. Clearing SWRST releases the USART for operation. See also chapter *USART Module, I2C mode* for USART0 when reconfiguring from I<sup>2</sup>C mode to SPI mode.

#### **Note: Initializing or Re-Configuring the USART Module**

The required USART initialization/re-configuration process is:

- 1) Set SWRST (`BIS.B #SWRST, &UxCTL`)
- 2) Initialize all USART registers with SWRST=1 (including UxCTL)
- 3) Enable USART module via the MEx SFRs (USPIE<sub>x</sub>)
- 4) Clear SWRST via software (`BIC.B #SWRST, &UxCTL`)
- 5) Enable interrupts (optional) via the IEx SFRs (URXIE<sub>x</sub> and/or UTXIE<sub>x</sub>)

Failure to follow this process may result in unpredictable USART behavior.

### 15.2.2 Master Mode

Figure 15–2. USART Master and External Slave

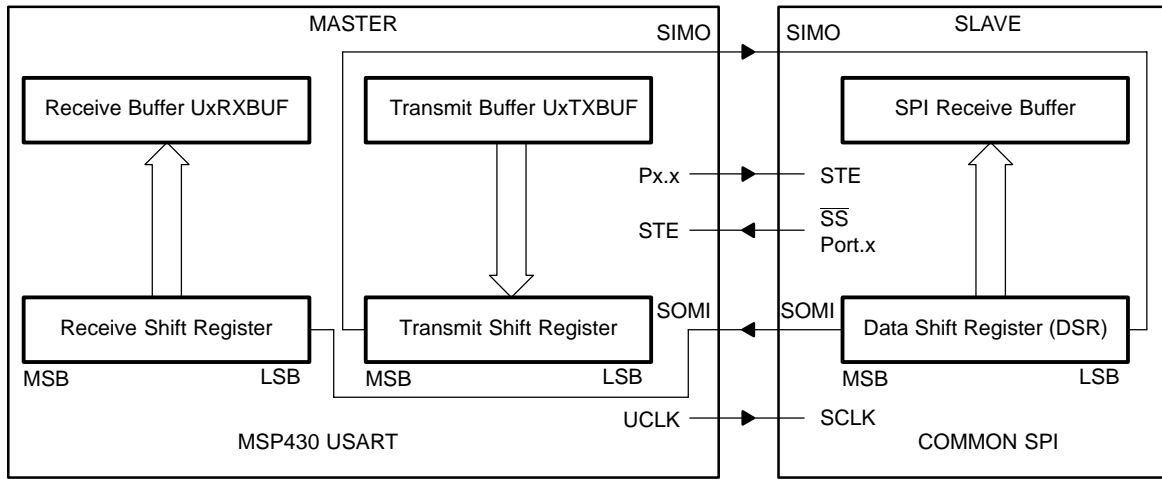


Figure 15–2 shows the USART as a master in both 3-pin and 4-pin configurations. The USART initiates data transfer when data is moved to the transmit data buffer UxTXBUF. The UxTXBUF data is moved to the TX shift register when the TX shift register is empty, initiating data transfer on SIMO starting with the most-significant bit. Data on SOMI is shifted into the receive shift register on the opposite clock edge, starting with the most-significant bit. When the character is received, the receive data is moved from the RX shift register to the received data buffer UxRXBUF and the receive interrupt flag, URXIFGx, is set, indicating the RX/TX operation is complete.

A set transmit interrupt flag, UTXIFGx, indicates that data has moved from UxTXBUF to the TX shift register and UxTXBUF is ready for new data. It does not indicate RX/TX completion.

To receive data into the USART in master mode, data must be written to UxTXBUF because receive and transmit operations operate concurrently.

#### Four-Pin SPI Master Mode

In 4-pin master mode, STE is used to prevent conflicts with another master. The master operates normally when STE is high. When STE is low:

- SIMO and UCLK are set to inputs and no longer drive the bus
- The error bit FE is set indicating a communication integrity violation to be handled by the user

A low STE signal does not reset the USART module. The STE input signal is not used in 3-pin master mode.

### 15.2.3 Slave Mode

Figure 15–3. USART Slave and External Master

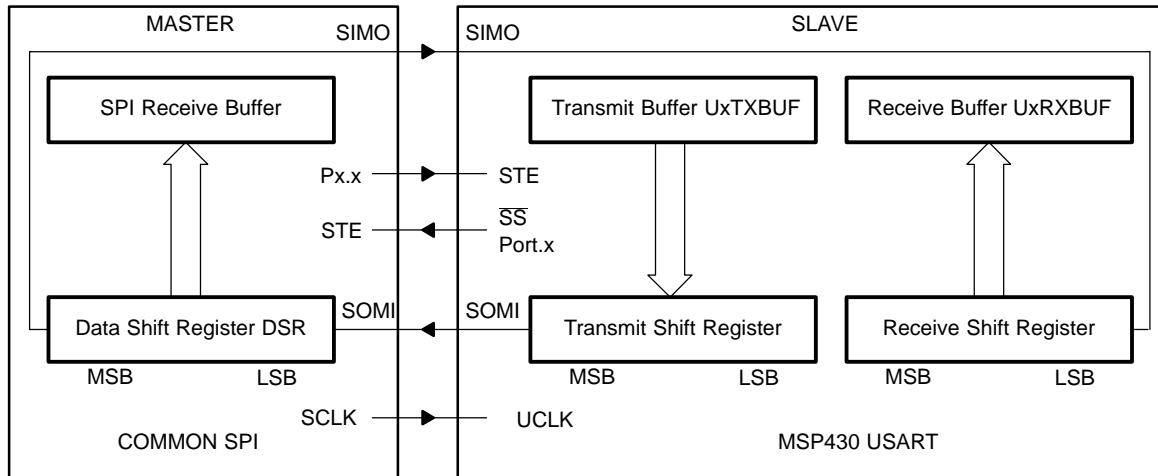


Figure 15–3 shows the USART as a slave in both 3-pin and 4-pin configurations. UCLK is used as the input for the SPI clock and must be supplied by the external master. The data-transfer rate is determined by this clock and not by the internal baud rate generator. Data written to UxTXBUF and moved to the TX shift register before the start of UCLK is transmitted on SOMI. Data on SIMO is shifted into the receive shift register on the opposite edge of UCLK and moved to UxRXBUF when the set number of bits are received. When data is moved from the RX shift register to UxRXBUF, the URXIFGx interrupt flag is set, indicating that data has been received. The overrun error bit, OE, is set when the previously received data is not read from UxRXBUF before new data is moved to UxRXBUF.

#### Four-Pin SPI Slave Mode

In 4-pin slave mode, STE is used by the slave to enable the transmit and receive operations and is provided by the SPI master. When STE is low, the slave operates normally. When STE is high:

- Any receive operation in progress on SIMO is halted
- SOMI is set to the input direction

A high STE signal does not reset the USART module. The STE input signal is not used in 3-pin slave mode.

### 15.2.4 SPI Enable

The SPI transmit/receive enable bit USPIEx enables or disables the USART in SPI mode. When USPIEx = 0, the USART stops operation after the current transfer completes, or immediately if no operation is active. A PUC or set SWRST bit disables the USART immediately and any active transfer is terminated.

#### Transmit Enable

When USPIEx = 0, any further write to UxTXBUF does not transmit. Data written to UxTXBUF will begin to transmit when USPIEx = 1 and the BRCLK source is active. Figure 15–4 and Figure 15–5 show the transmit enable state diagrams.

Figure 15–4. Master Mode Transmit Enable

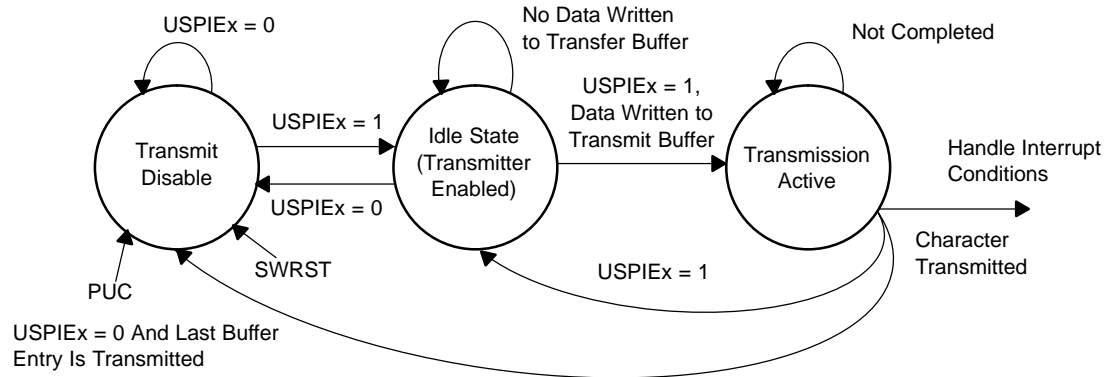
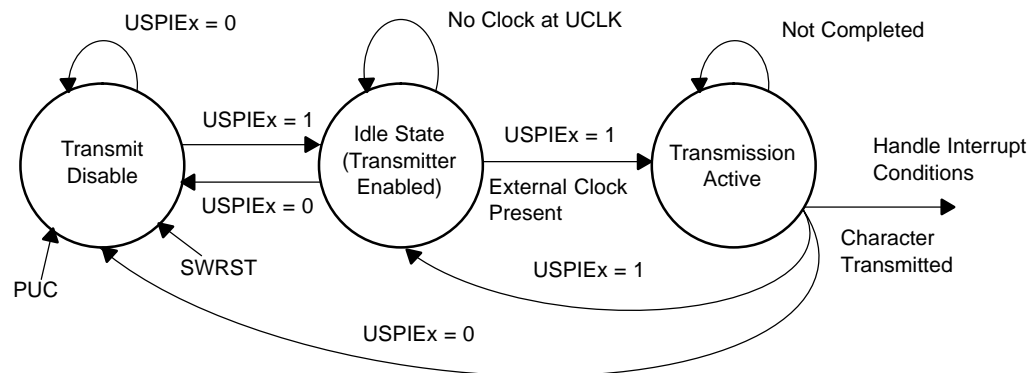


Figure 15–5. Slave Transmit Enable State Diagram



## Receive Enable

The SPI receive enable state diagrams are shown in Figure 15–6 and Figure 15–7. When USPIEx = 0, UCLK is disabled from shifting data into the RX shift register.

Figure 15–6. SPI Master Receive-Enable State Diagram

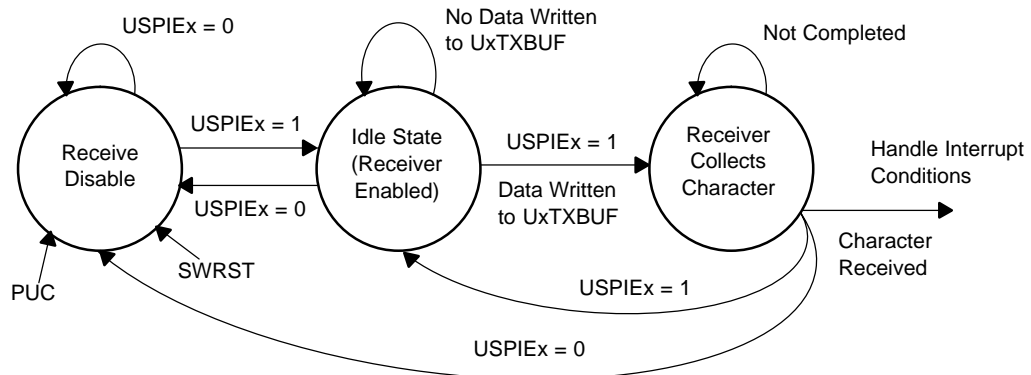
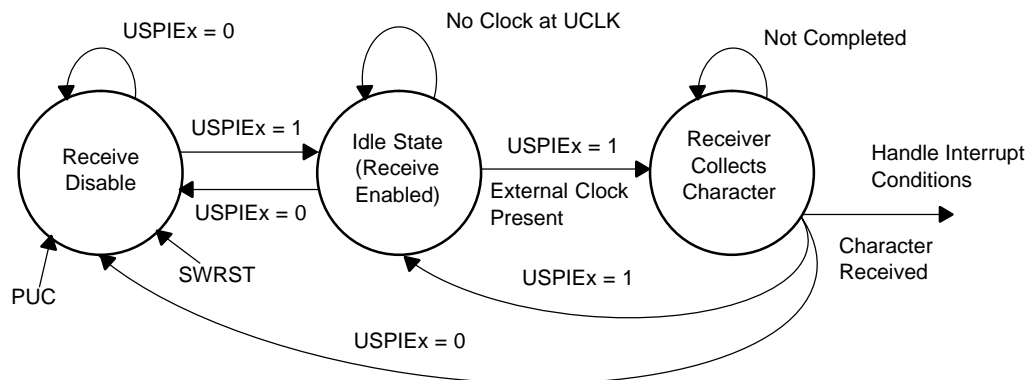


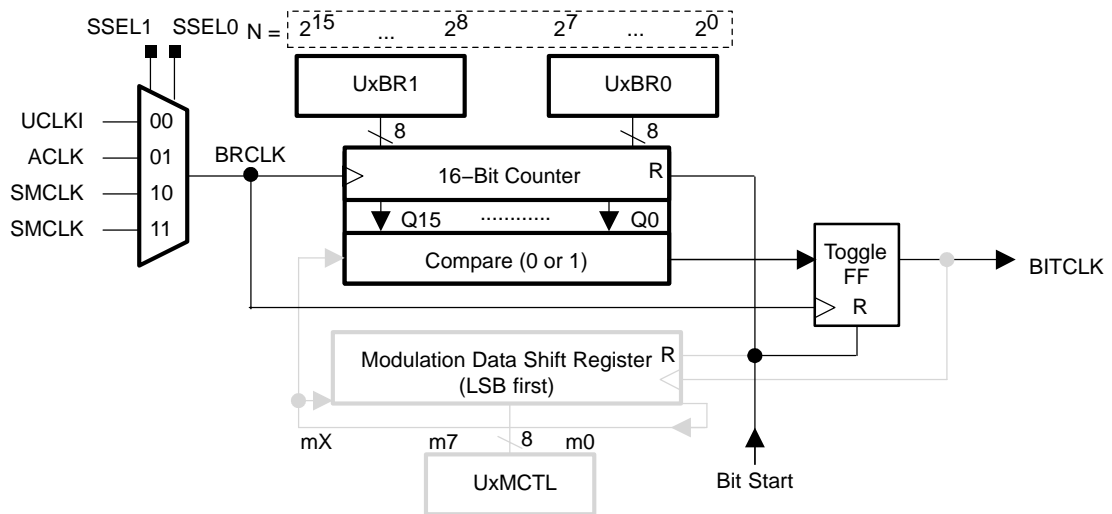
Figure 15–7. SPI Slave Receive-Enable State Diagram



### 15.2.5 Serial Clock Control

UCLK is provided by the master on the SPI bus. When MM = 1, BITCLK is provided by the USART baud rate generator on the UCLK pin as shown in Figure 15–8. When MM = 0, the USART clock is provided on the UCLK pin by the master and, the baud rate generator is not used and the SSELx bits are don't care. The SPI receiver and transmitter operate in parallel and use the same clock source for data transfer.

Figure 15–8. SPI Baud Rate Generator



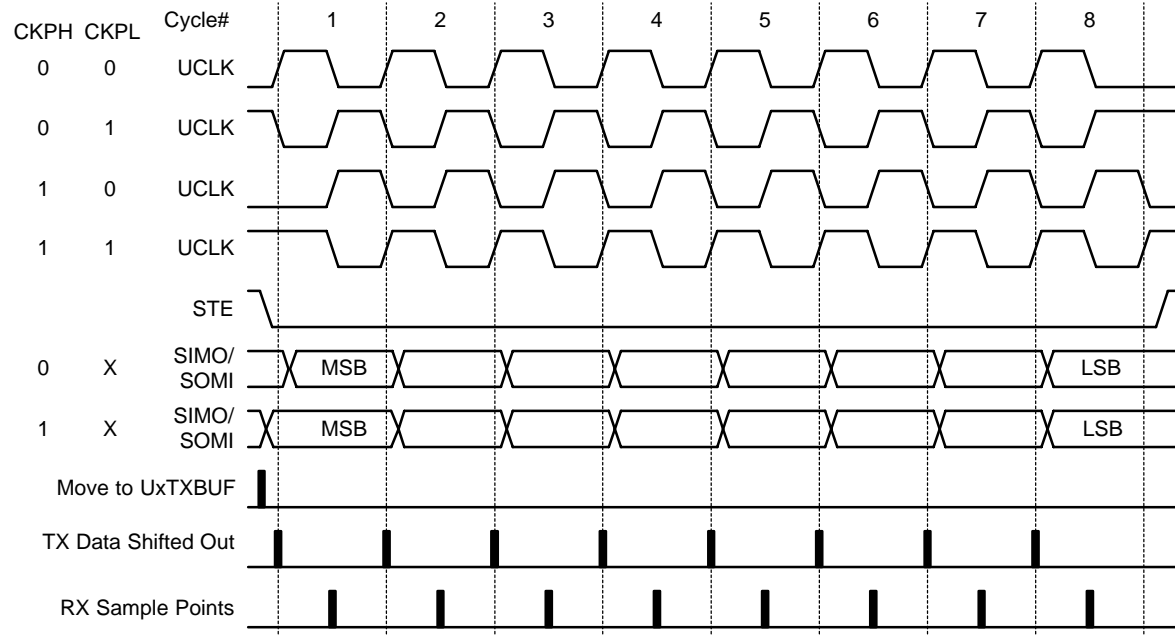
The 16-bit value of UxBR0+UxBR1 is the division factor of the USART clock source, BRCLK. The maximum baud rate that can be generated in master mode is BRCLK/2. The maximum baud rate that can be generated in slave mode is BRCLK. The modulator in the USART baud rate generator is not used for SPI mode and is recommended to be set to 000h. The UCLK frequency is given by:

$$\text{Baud rate} = \frac{BRCLK}{UxBR} \text{ with } UxBR = [UxBR1, UxBR0]$$

### Serial Clock Polarity and Phase

The polarity and phase of UCLK are independently configured via the CKPL and CKPH control bits of the USART. Timing for each case is shown in Figure 15–9.

Figure 15–9. USART SPI Timing





### 15.2.6 SPI Interrupts

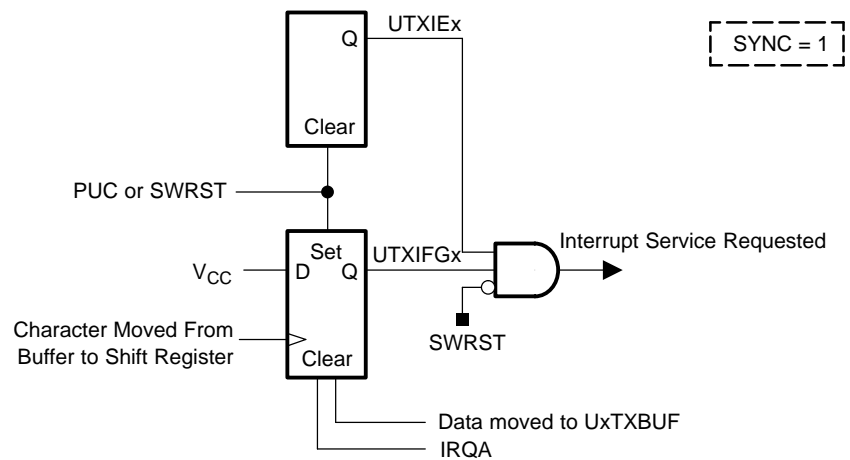
The USART has one interrupt vector for transmission and one interrupt vector for reception.

#### SPI Transmit Interrupt Operation

The UTXIFGx interrupt flag is set by the transmitter to indicate that UxTXBUF is ready to accept another character. An interrupt request is generated if UTXIEx and GIE are also set. UTXIFGx is automatically reset if the interrupt request is serviced or if a character is written to UxTXBUF.

UTXIFGx is set after a PUC or when SWRST = 1. UTXIEx is reset after a PUC or when SWRST = 1. The operation is shown in Figure 15–10.

Figure 15–10. Transmit Interrupt Operation



**Note: Writing to UxTXBUF in SPI Mode**  
 Data written to UxTXBUF when UTXIFGx = 0 and USPIEx = 1 may result in erroneous data transmission.

### SPI Receive Interrupt Operation

The URXIFGx interrupt flag is set each time a character is received and loaded into UxRXBUF as shown in Figure 15–11 and Figure 15–12. An interrupt request is generated if URXIE<sub>x</sub> and GIE are also set. URXIFG<sub>x</sub> and URXIE<sub>x</sub> are reset by a system reset PUC signal or when SWRST = 1. URXIFG<sub>x</sub> is automatically reset if the pending interrupt is served or when UxRXBUF is read.

Figure 15–11. Receive Interrupt Operation

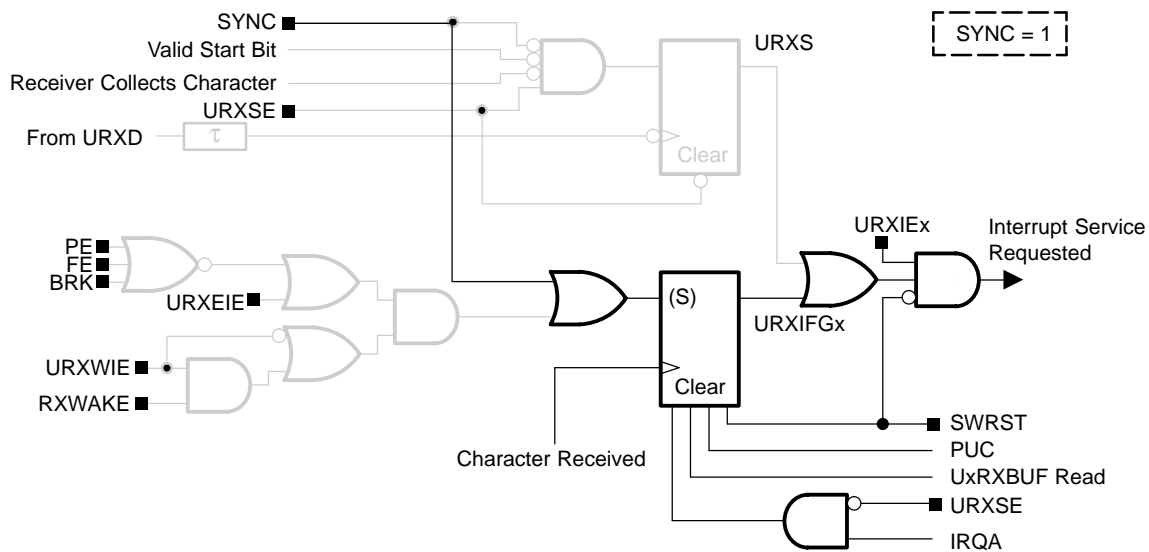
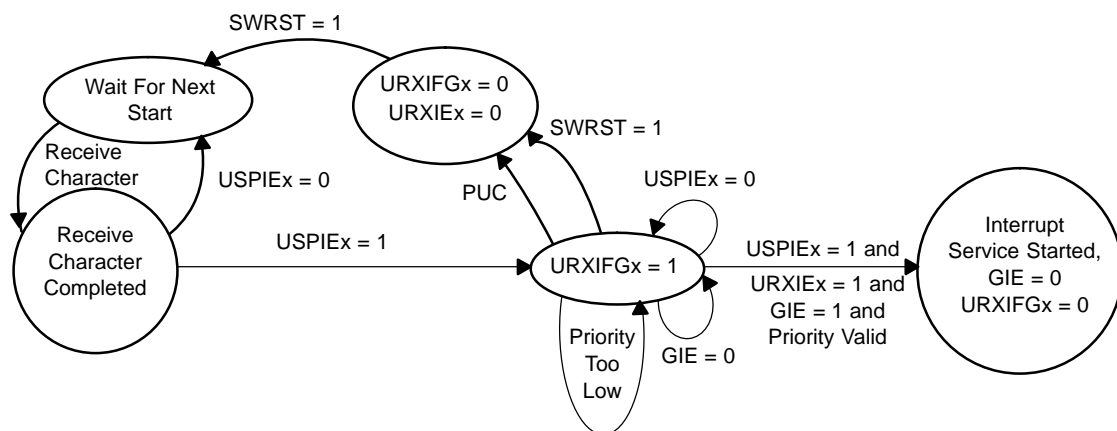


Figure 15–12. Receive Interrupt State Diagram



### 15.3 USART Registers: SPI Mode

Table 15–1 lists the registers for all devices implementing a USART module. Table 15–2 applies only to devices with a second USART module, USART1.

Table 15–1. USART0 Control and Status Registers

Register	Short Form	Register Type	Address	Initial State
USART control register	U0CTL	Read/write	070h	001h with PUC
Transmit control register	U0TCTL	Read/write	071h	001h with PUC
Receive control register	U0RCTL	Read/write	072h	000h with PUC
Modulation control register	U0MCTL	Read/write	073h	Unchanged
Baud rate control register 0	U0BR0	Read/write	074h	Unchanged
Baud rate control register 1	U0BR1	Read/write	075h	Unchanged
Receive buffer register	U0RXBUF	Read	076h	Unchanged
Transmit buffer register	U0TXBUF	Read/write	077h	Unchanged
SFR module enable register 1	ME1	Read/write	004h	000h with PUC
SFR interrupt enable register 1	IE1	Read/write	000h	000h with PUC
SFR interrupt flag register 1	IFG1	Read/write	002h	082h with PUC

Table 15–2. USART1 Control and Status Registers

Register	Short Form	Register Type	Address	Initial State
USART control register	U1CTL	Read/write	078h	001h with PUC
Transmit control register	U1TCTL	Read/write	079h	001h with PUC
Receive control register	U1RCTL	Read/write	07Ah	000h with PUC
Modulation control register	U1MCTL	Read/write	07Bh	Unchanged
Baud rate control register 0	U1BR0	Read/write	07Ch	Unchanged
Baud rate control register 1	U1BR1	Read/write	07Dh	Unchanged
Receive buffer register	U1RXBUF	Read	07Eh	Unchanged
Transmit buffer register	U1TXBUF	Read/write	07Fh	Unchanged
SFR module enable register 2	ME2	Read/write	005h	000h with PUC
SFR interrupt enable register 2	IE2	Read/write	001h	000h with PUC
SFR interrupt flag register 2	IFG2	Read/write	003h	020h with PUC

**Note: Modifying the SFR bits**

To avoid modifying control bits for other modules, it is recommended to set or clear the IEx and IFGx bits using `BIS . B` or `BIC . B` instructions, rather than `MOV . B` or `CLR . B` instructions.

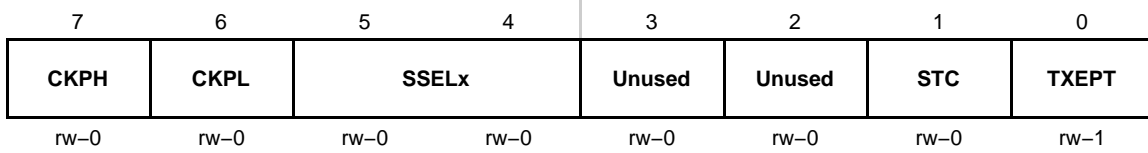
## UxCTL, USART Control Register

7	6	5	4	3	2	1	0
Unused	Unused	I2C <sup>†</sup>	CHAR	LISTEN	SYNC	MM	SWRST
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1

<b>Unused</b>	Bits 7-6	Unused
<b>I2C<sup>†</sup></b>	Bit 5	I2C mode enable. This bit selects I2C or SPI operation when SYNC = 1. 0 SPI mode 1 I <sup>2</sup> C mode
<b>CHAR</b>	Bit 4	Character length 0 7-bit data 1 8-bit data
<b>LISTEN</b>	Bit 3	Listen enable. The LISTEN bit selects the loopback mode 0 Disabled 1 Enabled. The transmit signal is internally fed back to the receiver
<b>SYNC</b>	Bit 2	Synchronous mode enable 0 UART mode 1 SPI mode
<b>MM</b>	Bit 1	Master mode 0 USART is slave 1 USART is master
<b>SWRST</b>	Bit 0	Software reset enable 0 Disabled. USART reset released for operation 1 Enabled. USART logic held in reset state

<sup>†</sup> Applies to USART0 on MSP430x15x and MSP430x16x devices only.

**UxTCTL, USART Transmit Control Register**



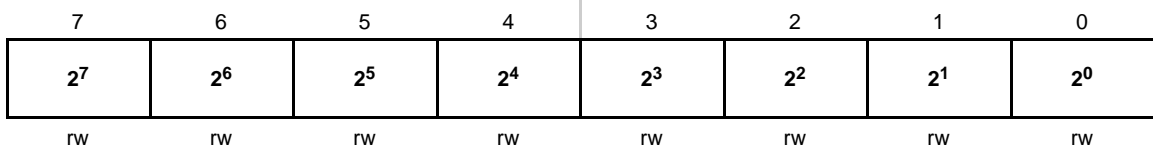
<b>CKPH</b>	Bit 7	Clock phase select. Controls the phase of UCLK. 0 Normal UCLK clocking scheme 1 UCLK is delayed by one half cycle
<b>CKPL</b>	Bit 6	Clock polarity select 0 The inactive level is low; data is output with the rising edge of UCLK; input data is latched with the falling edge of UCLK. 1 The inactive level is high; data is output with the falling edge of UCLK; input data is latched with the rising edge of UCLK.
<b>SSELx</b>	Bits 5-4	Source select. These bits select the BRCLK source clock. 00 External UCLK (valid for slave mode only) 01 ACLK (valid for master mode only) 10 SMCLK (valid for master mode only) 11 SMCLK (valid for master mode only)
<b>Unused</b>	Bit 3	Unused
<b>Unused</b>	Bit 2	Unused
<b>STC</b>	Bit 1	Slave transmit control. 0 4-pin SPI mode: STE enabled. 1 3-pin SPI mode: STE disabled.
<b>TXEPT</b>	Bit 0	Transmitter empty flag. The TXEPT flag is not used in slave mode. 0 Transmission active and/or data waiting in UxTXBUF 1 UxTXBUF and TX shift register are empty

### UxRCTL, USART Receive Control Register

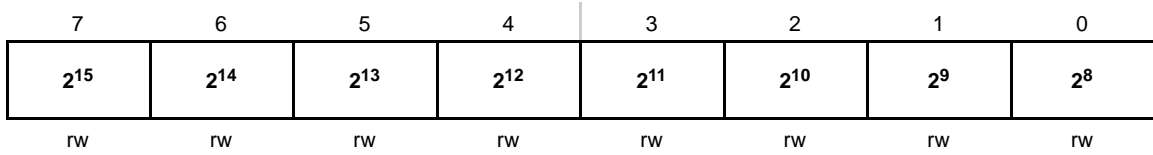
7	6	5	4	3	2	1	0
<b>FE</b>	<b>Unused</b>	<b>OE</b>	<b>Unused</b>	<b>Unused</b>	<b>Unused</b>	<b>Unused</b>	<b>Unused</b>
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

<b>FE</b>	Bit 7	Framing error flag. This bit indicates a bus conflict when MM = 1 and STC = 0. FE is unused in slave mode. 0 No conflict detected 1 A negative edge occurred on STE, indicating bus conflict
<b>Undefined</b>	Bit 6	Unused
<b>OE</b>	Bit 5	Overrun error flag. This bit is set when a character is transferred into UxRXBUF before the previous character was read. OE is automatically reset when UxRXBUF is read, when SWRST = 1, or can be reset by software. 0 No error 1 Overrun error occurred
<b>Unused</b>	Bit 4	Unused
<b>Unused</b>	Bit 3	Unused
<b>Unused</b>	Bit 2	Unused
<b>Unused</b>	Bit 1	Unused
<b>Unused</b>	Bit 0	Unused

**UxBR0, USART Baud Rate Control Register 0**

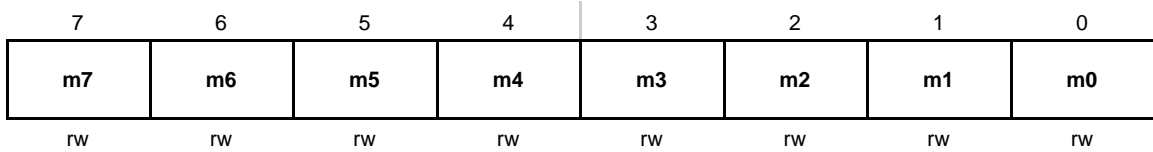


**UxBR1, USART Baud Rate Control Register 1**



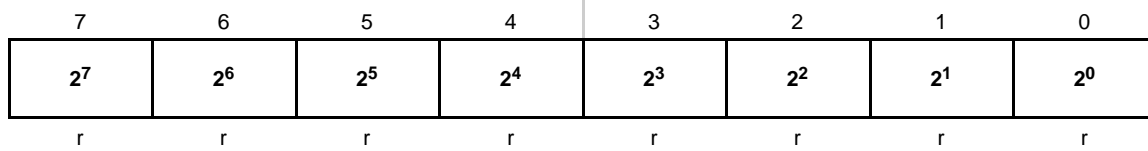
**UxBRx**                      The baud-rate generator uses the content of {UxBR1+UxBR0} to set the baud rate. Unpredictable SPI operation occurs if UxBR < 2.

**UxMCTL, USART Modulation Control Register**



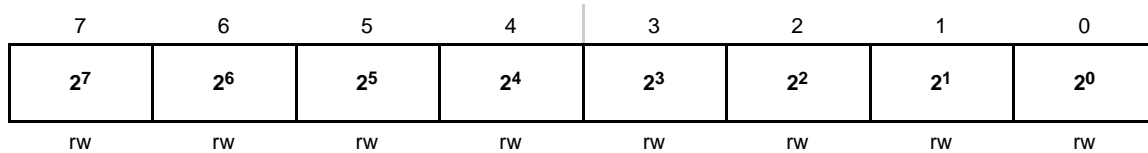
**UxMCTLx**      Bits      The modulation control register is not used for SPI mode and should be set to 000h.  
                      7–0

### UxRXBUF, USART Receive Buffer Register



**UxRXBUFx** Bits 7–0      The receive-data buffer is user accessible and contains the last received character from the receive shift register. Reading UxRXBUF resets the OE bit and URXIFGx flag. In 7-bit data mode, UxRXBUF is LSB justified and the MSB is always reset.

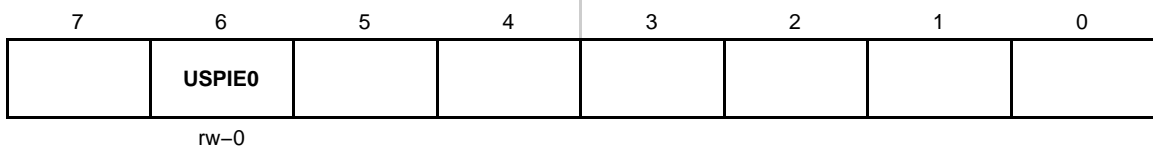
### UxTXBUF, USART Transmit Buffer Register



**UxTXBUFx** Bits 7–0      The transmit data buffer is user accessible and contains current data to be transmitted. When seven-bit character-length is used, the data should be MSB justified before being moved into UxTXBUF. Data is transmitted MSB first. Writing to UxTXBUF clears UTXIFGx.

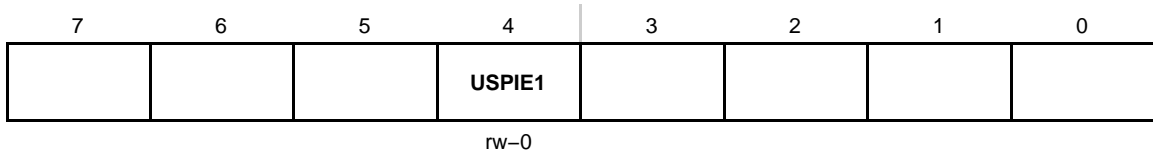


**ME1, Module Enable Register 1**



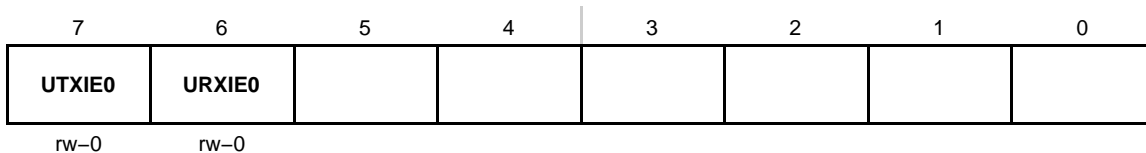
- Bit 7     This bit may be used by other modules. See device-specific datasheet.
- USPIE0**   Bit 6     USART0 SPI enable. This bit enables the SPI mode for USART0.
  - 0     Module not enabled
  - 1     Module enabled
- Bits 5-0     These bits may be used by other modules. See device-specific datasheet.

**ME2, Module Enable Register 2**



- Bits 7-5     These bits may be used by other modules. See device-specific datasheet.
- USPIE1**   Bit 4     USART1 SPI enable. This bit enables the SPI mode for USART1.
  - 0     Module not enabled
  - 1     Module enabled
- Bits 3-0     These bits may be used by other modules. See device-specific datasheet.

### IE1, Interrupt Enable Register 1

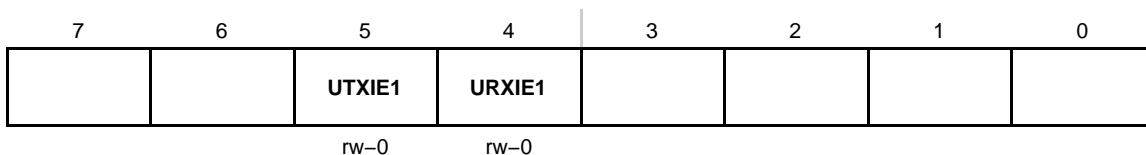


**UTXIE0** Bit 7 USART0 transmit interrupt enable. This bit enables the UTXIFG0 interrupt.  
 0 Interrupt not enabled  
 1 Interrupt enabled

**URXIE0** Bit 6 USART0 receive interrupt enable. This bit enables the URXIFG0 interrupt.  
 0 Interrupt not enabled  
 1 Interrupt enabled

Bits 5-0 These bits may be used by other modules. See device-specific datasheet.

### IE2, Interrupt Enable Register 2



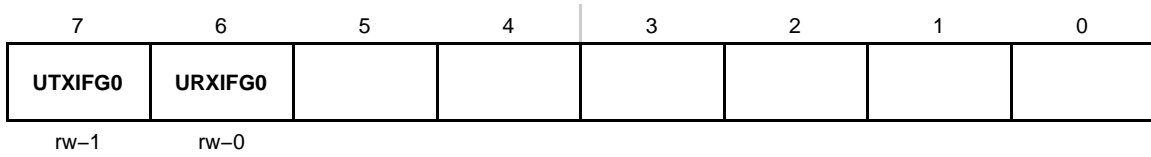
Bits 7-6 These bits may be used by other modules. See device-specific datasheet.

**UTXIE1** Bit 5 USART1 transmit interrupt enable. This bit enables the UTXIFG1 interrupt.  
 0 Interrupt not enabled  
 1 Interrupt enabled

**URXIE1** Bit 4 USART1 receive interrupt enable. This bit enables the URXIFG1 interrupt.  
 0 Interrupt not enabled  
 1 Interrupt enabled

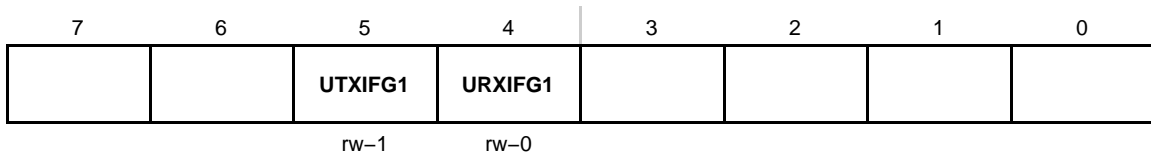
Bits 3-0 These bits may be used by other modules. See device-specific datasheet.

### IFG1, Interrupt Flag Register 1



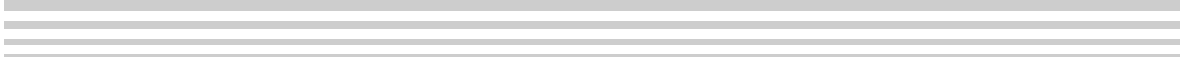
- UTXIFG0** Bit 7 USART0 transmit interrupt flag. UTXIFG0 is set when U0TXBUF is empty.  
 0 No interrupt pending  
 1 Interrupt pending
- URXIFG0** Bit 6 USART0 receive interrupt flag. URXIFG0 is set when U0RXBUF has received a complete character.  
 0 No interrupt pending  
 1 Interrupt pending
- Bits 5-0 These bits may be used by other modules. See device-specific datasheet.

### IFG2, Interrupt Flag Register 2



- Bits 7-6 These bits may be used by other modules. See device-specific datasheet.
- UTXIFG1** Bit 5 USART1 transmit interrupt flag. UTXIFG1 is set when U1TXBUF is empty.  
 0 No interrupt pending  
 1 Interrupt pending
- URXIFG1** Bit 4 USART1 receive interrupt flag. URXIFG1 is set when U1RXBUF has received a complete character.  
 0 No interrupt pending  
 1 Interrupt pending
- Bits 3-0 These bits may be used by other modules. See device-specific datasheet.

# OA



The OA is a general purpose operational amplifier. This chapter describes the OA. Three OA modules are implemented in the MSP430FG43x devices.

<b>Topic</b>	<b>Page</b>
<b>16.1 OA Introduction</b> .....	<b>16-2</b>
<b>16.2 OA Operation</b> .....	<b>16-4</b>
<b>16.3 OA Registers</b> .....	<b>16-11</b>

## 16.1 OA Introduction

The OA op amps support front-end analog signal conditioning prior to analog-to-digital conversion.

Features of the OA include:

- Single supply, low-current operation
- Rail-to-rail output
- Software selectable Rail-to-Rail input
- Programmable settling time vs. power consumption
- Software selectable configurations
- Software selectable feedback resistor ladder for PGA implementations

---

**Note: Multiple OA Modules**

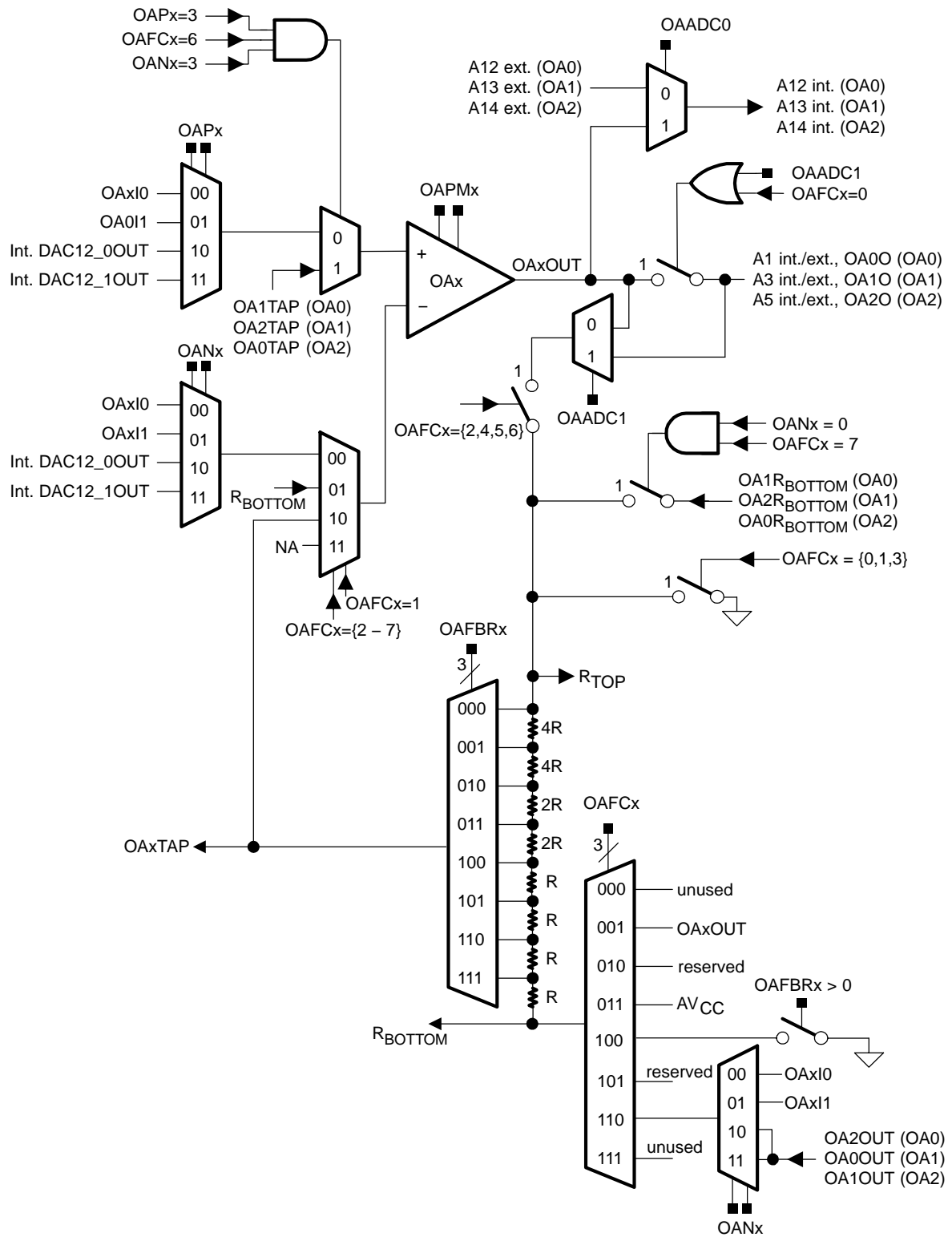
Some devices may integrate more than one OA module. In the case where more than one OA is present on a device, the multiple OA modules operate identically.

Throughout this chapter, nomenclature appears such as OAxCTL0 to describe register names. When this occurs, the x is used to indicate which OA module is being discussed. In cases where operation is identical, the register is simply referred to as OAxCTL0.

---

The block diagram of the OA module is shown in Figure 16–1.

Figure 16–1. OA Block Diagram



## 16.2 OA Operation

The OA module is configured with user software. The setup and operation of the OA is discussed in the following sections.

### 16.2.1 OA Amplifier

The OA is a configurable, low-current, rail-to-rail operational amplifier. It can be configured as an inverting amplifier, or a non-inverting amplifier, or can be combined with other OA modules to form differential amplifiers. The output slew rate of the OA can be configured for optimized settling time vs. power consumption with the OAPMx bits. When OAPMx = 00 the OA is off and the output is high-impedance. When OAPMx > 0, the OA is on. See the device-specific datasheet for parameters.

### 16.2.2 OA Input

The OA has configurable input selection. The signals for the + and – inputs are individually selected with the OANx and OAPx bits and can be selected as external signals or internal signals from one of the DAC12 modules. One of the non-inverting inputs is tied together internally for all OA modules.

The OA input signal swing is software selectable with the OARRIP bit. When OARRIP = 0, rail-to-rail input mode is selected and the OA uses higher quiescent current. See the device datasheet for parameters.

### 16.2.3 OA Output

The OA has configurable output selection. The OA output signals can be routed to ADC12 inputs A12 (OA0), A13 (OA1), or A14 (OA2) with the OAADC0 bit. When OAADC0 = 1 and OAPMx > 0, the OA output is connected to the corresponding ADC input internally, and the external ADC input is not connected. The OA output signals can also be routed to ADC12 inputs A1 (OA0), A3 (OA1), or A5 (OA2) when OAFcx = 0, or when OAADC1 = 1. In this case, the OA output is connected to both the ADC12 input internally, and the corresponding pin on the device. The OA output is also connected to an internal R-ladder with the OAFcx bits. The R-ladder tap is selected with the OAFBRx bits to provide programmable gain amplifier functionality.

## 16.2.4 OA Configurations

The OA can be configured for different amplifier functions with the OAF<sub>Cx</sub> bits, as listed in Table 16–1.

Table 16–1. OA Mode Select

OAF <sub>Cx</sub>	OA Mode
000	General-purpose opamp
001	Unity gain buffer
010	Reserved
011	Comparator
100	Non-inverting PGA amplifier
101	Reserved
110	Inverting PGA amplifier
111	Differential amplifier

### General Purpose Opamp Mode

In this mode the feedback resistor ladder is isolated from the O<sub>Ax</sub> and the O<sub>Ax</sub>CTL0 bits define the signal routing. The O<sub>Ax</sub> inputs are selected with the OAP<sub>x</sub> and OAN<sub>x</sub> bits. The O<sub>Ax</sub> output is internally connected to the ADC12 input channel as selected by the O<sub>Ax</sub>CTL0 bits.

### Unity Gain Mode

In this mode the output of the O<sub>Ax</sub> is connected to R<sub>BOTTOM</sub> and the inverting input of the O<sub>Ax</sub> providing a unity gain buffer. The non-inverting input is selected by the OAP<sub>x</sub> bits. The external connection for the inverting input is disabled and the OAN<sub>x</sub> bits are don't care. The O<sub>Ax</sub> output is internally connected to the ADC12 input channel as selected by the O<sub>Ax</sub>CTL0 bits.

### Comparator Mode

In this mode the output of the O<sub>Ax</sub> is isolated from the resistor ladder. R<sub>TOP</sub> is connected to AV<sub>SS</sub> and R<sub>BOTTOM</sub> is connected to AV<sub>CC</sub>. The O<sub>Ax</sub>TAP signal is connected to the inverting input of the O<sub>Ax</sub> providing a comparator with a programmable threshold voltage selected by the OAFBR<sub>x</sub> bits. The non-inverting input is selected by the OAP<sub>x</sub> bits. Hysteresis can be added by an external positive feedback resistor. The external connection for the inverting input is disabled and the OAN<sub>x</sub> bits are don't care. The O<sub>Ax</sub> output is internally connected to the ADC12 input channel as selected by the O<sub>Ax</sub>CTL0 bits.



### Non-Inverting PGA Mode

In this mode the output of the OAx is connected to  $R_{TOP}$  and  $R_{BOTTOM}$  is connected to  $AV_{SS}$ . The OAxTAP signal is connected to the inverting input of the OAx providing a non-inverting amplifier configuration with a programmable gain of  $[1+OAxTAP \text{ ratio}]$ . The OAxTAP ratio is selected by the OAFBRx bits. If the OAFBRx bits = 0, the gain is unity. The non-inverting input is selected by the OAPx bits. The external connection for the inverting input is disabled and the OANx bits are don't care. The OAx output is internally connected to the ADC12 input channel as selected by the OAxCTL0 bits.

### Inverting PGA Mode

In this mode the output of the OAx is connected to  $R_{TOP}$  and  $R_{BOTTOM}$  is connected to an analog multiplexer that multiplexes the OAxI0, OAxI1 or the output of one of the remaining OAs, selected with the OANx bits. The OAxTAP signal is connected to the inverting input of the OAx providing an inverting amplifier with a gain of  $-OAxTAP \text{ ratio}$ . The OAxTAP ratio is selected by the OAFBRx bits. The non-inverting input is selected by the OAPx bits. The OAx output is internally connected to the ADC12 input channel as selected by the OAxCTL0 bits.

### Differential Amplifier Mode

This mode allows internal routing of the OA signals for a two-opamp or three-opamp instrumentation amplifier. Figure 16–2 shows a two-opamp configuration with OA0 and OA1. In this mode the output of the OAx is connected to  $R_{TOP}$  by routing through another OAx in the Inverting PGA mode.  $R_{BOTTOM}$  is unconnected providing a unity gain buffer. This buffer is combined with one or two remaining OAx to form the differential amplifier. The OAx output is internally connected to the ADC12 input channel as selected by the OAxCTL0 bits.

Figure 16–2 shows an example of a two-opamp differential amplifier using OA0 and OA1. The control register settings and are shown in Table 16–2. The gain for the amplifier is selected by the OAFBRx bits for OA1 and is shown in Table 16–3. The OAx interconnections are shown in Figure 16–3.

Table 16–2. Two-Opamp Differential Amplifier Control Register Settings

Register	Settings (binary)
OA0CTL0	00 xx xx 0 0
OA0CTL1	000 111 0 x
OA1CTL0	10 xx xx x x
OA1CTL1	xxx 110 0 x

Table 16–3. Two-Opamp Differential Amplifier Gain Settings

OA1 OAFBRx	Gain
000	0
001	1/3
010	1
011	1 2/3
100	3
101	4 1/3
110	7
111	15

Figure 16–2. Two Opamp Differential Amplifier

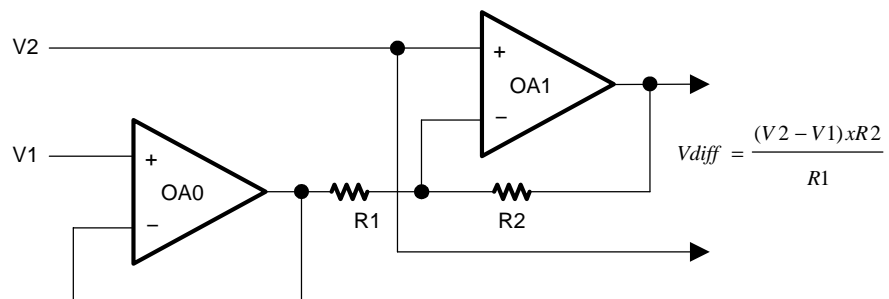


Figure 16–3. Two Opamp Differential Amplifier OAx Interconnections

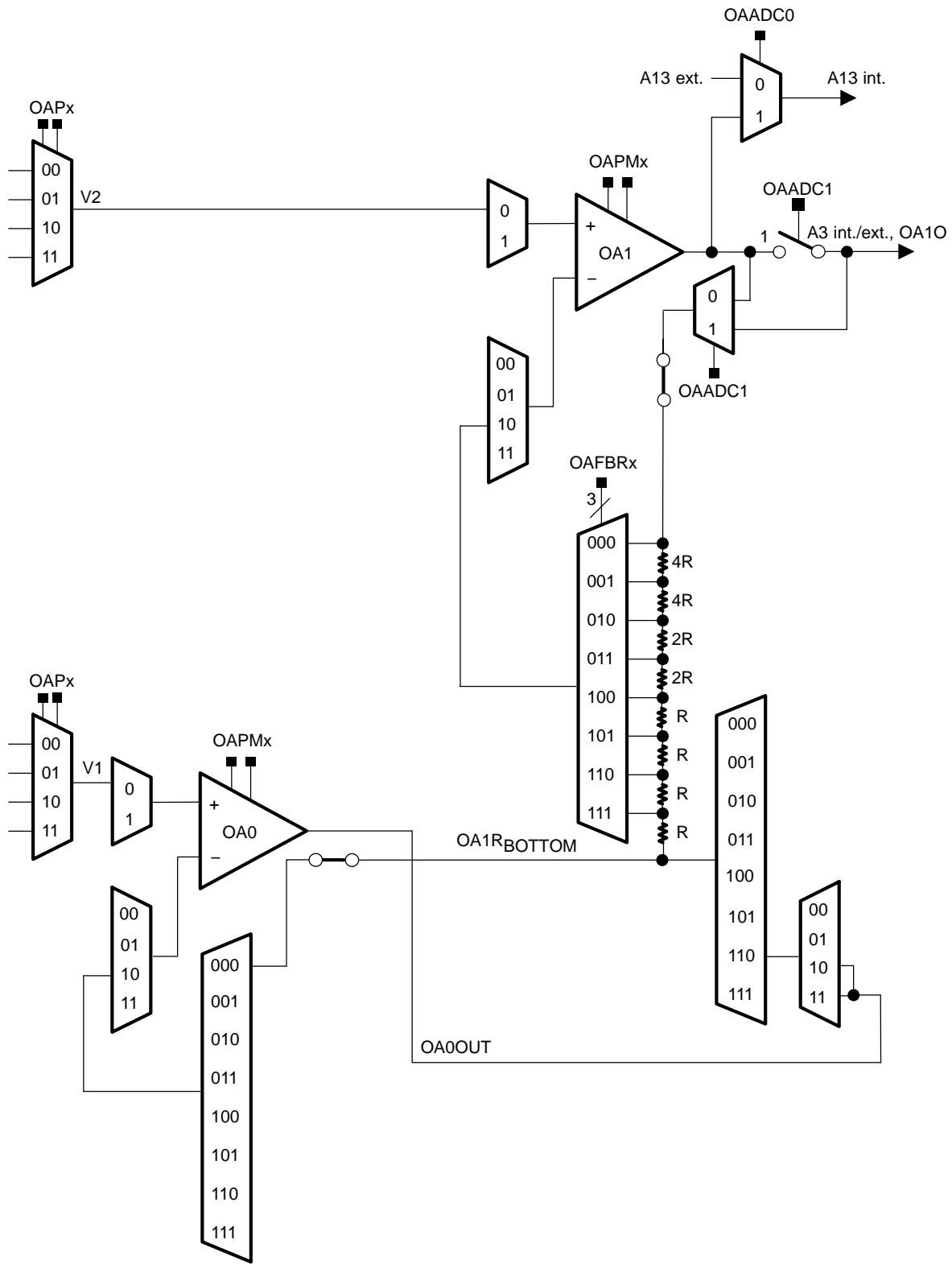


Figure 16–4 shows an example of a three-opamp differential amplifier using OA0, OA1 and OA2. The control register settings are shown in Table 16–4. The gain for the amplifier is selected by the OAFBRx bits of OA0 and OA2. The OAFBRx settings for both OA0 and OA2 must be equal. The gain settings are shown in Table 16–5. The OAx interconnections are shown in Figure 16–5.

Table 16–4. Three-Opamp Differential Amplifier Control Register Settings

Register	Settings (binary)
OA0CTL0	00 xx xx 0 0
OA0CTL1	xxx 001 0 x
OA1CTL0	00 xx xx 0 0
OA1CTL1	000 111 0 x
OA2CTL0	11 11 xx x x
OA2CTL1	xxx 110 0 x

Table 16–5. Three-Opamp Differential Amplifier Gain Settings

OA0/OA2 OAFBRx	Gain
000	0
001	1/3
010	1
011	1 2/3
100	3
101	4 1/3
110	7
111	15

Figure 16–4. Three Opamp Differential Amplifier

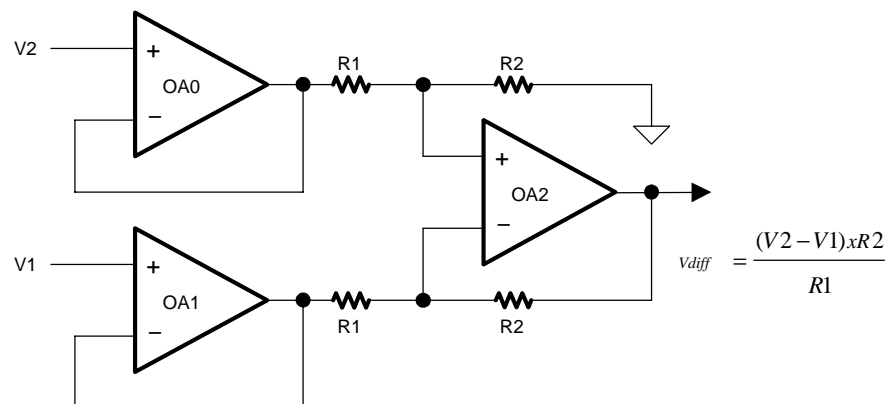
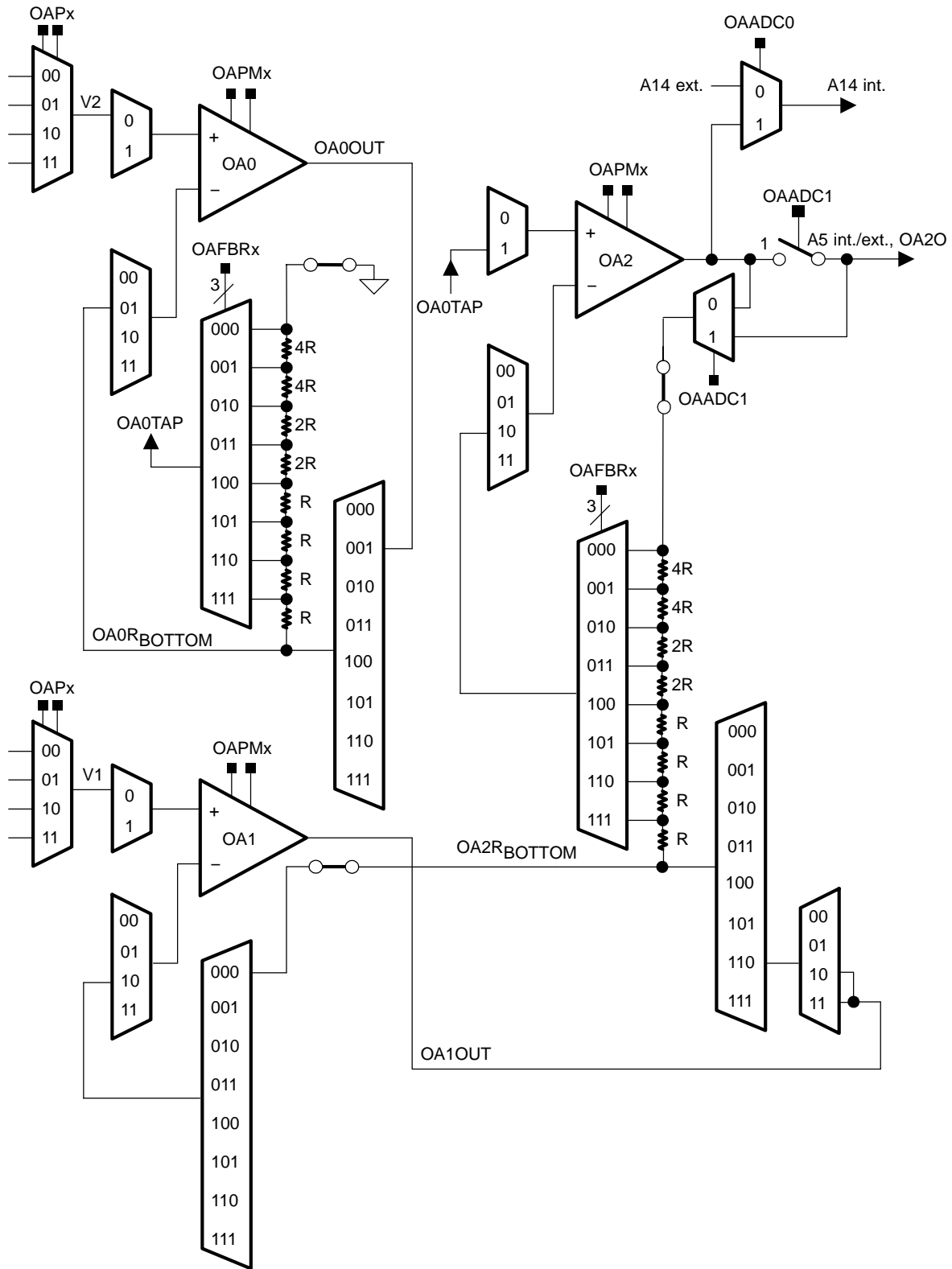


Figure 16–5. Three Opamp Differential Amplifier OAx Interconnections



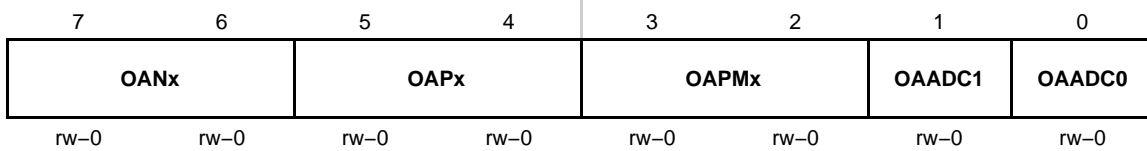
## 16.3 OA Registers

The OA registers are listed in Table 16–6.

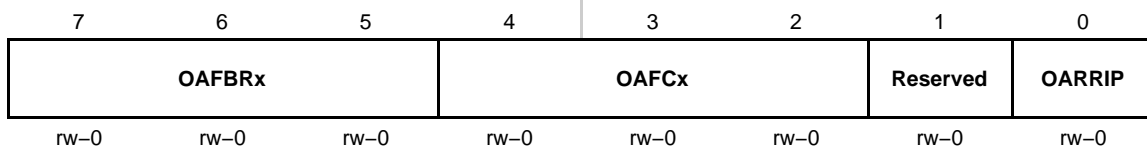
Table 16–6.

Register	Short Form	Register Type	Address	Initial State
OA0 Control Register 0	OA0CTL0	Read/write	0C0h	Reset with POR
OA0 Control Register 1	OA0CTL1	Read/write	0C1h	Reset with POR
OA1 Control Register 0	OA1CTL0	Read/write	0C2h	Reset with POR
OA1 Control Register 1	OA1CTL1	Read/write	0C3h	Reset with POR
OA2 Control Register 0	OA2CTL0	Read/write	0C4h	Reset with POR
OA2 Control Register 1	OA2CTL1	Read/write	0C5h	Reset with POR

**OAxCTL0, Opamp Control Register 0**



<b>OANx</b>	Bits 7-6	Inverting input select. These bits select the input signal for the OA inverting input. 00 OAxI0 01 OAxI1 10 DAC0 internal 11 DAC1 internal
<b>OAPx</b>	Bits 5-4	Non-inverting input select. These bits select the input signal for the OA non-inverting input. 00 OAxI0 01 OA0I1 10 DAC0 internal 11 DAC1 internal
<b>OAPMx</b>	Bits 3-2	Slew rate select These bits select the slew rate vs. current consumption for the OA. 00 Off, output high Z 01 Slow 10 Medium 11 Fast
<b>OAADC1</b>	Bit 1	OA output select. This bit connects the OAx output to ADC12 input Ax and output pin OAxO when OAFcx > 0. 0 OAx output not connected to A1 (OA0), A3 (OA1), or A5 (OA2) 1 OAx output connected to A1 (OA0), A3 (OA1), or A5 (OA2)
<b>OAADC0</b>	Bit 0	OA output select. This bit connects the OAx output to ADC12 input Ax when OAPMx > 0. 0 OAx output not connected to A12 (OA0), A13 (OA1), or A14 (OA2) 1 OAx output connected to A12 (OA0), A13 (OA1), or A14 (OA2)

**OAxCTL1, Opamp Control Register 1**

<b>OAFBRx</b>	Bits 7-5	OAx feedback resistor select 000 Tap 0 001 Tap 1 010 Tap 2 011 Tap 3 100 Tap 4 101 Tap 5 110 Tap 6 111 Tap 7
<b>O AFCx</b>	Bits 4-2	OAx function control. This bit selects the function of OAx 000 General purpose 001 Unity gain buffer 010 Reserved 011 Comparing amplifier 100 Non-inverting PGA 101 Reserved 110 Inverting PGA 111 Differential amplifier
<b>Reserved</b>	Bit 1	Reserved
<b>OARRIP</b>	Bit 0	OA rail-to-rail input off. 0 OAx input signal range is rail-to-rail 1 OAx input signal range is limited. See device datasheet for parameters.



# Comparator\_A

---

---

---

---

Comparator\_A is an analog voltage comparator. This chapter describes Comparator\_A. Comparator\_A is implemented in all MSP430x4xx devices.

<b>Topic</b>	<b>Page</b>
17.1 Comparator_A Introduction .....	17-2
17.2 Comparator_A Operation .....	17-4
17.3 Comparator_A Registers .....	17-9

## 17.1 Comparator\_A Introduction

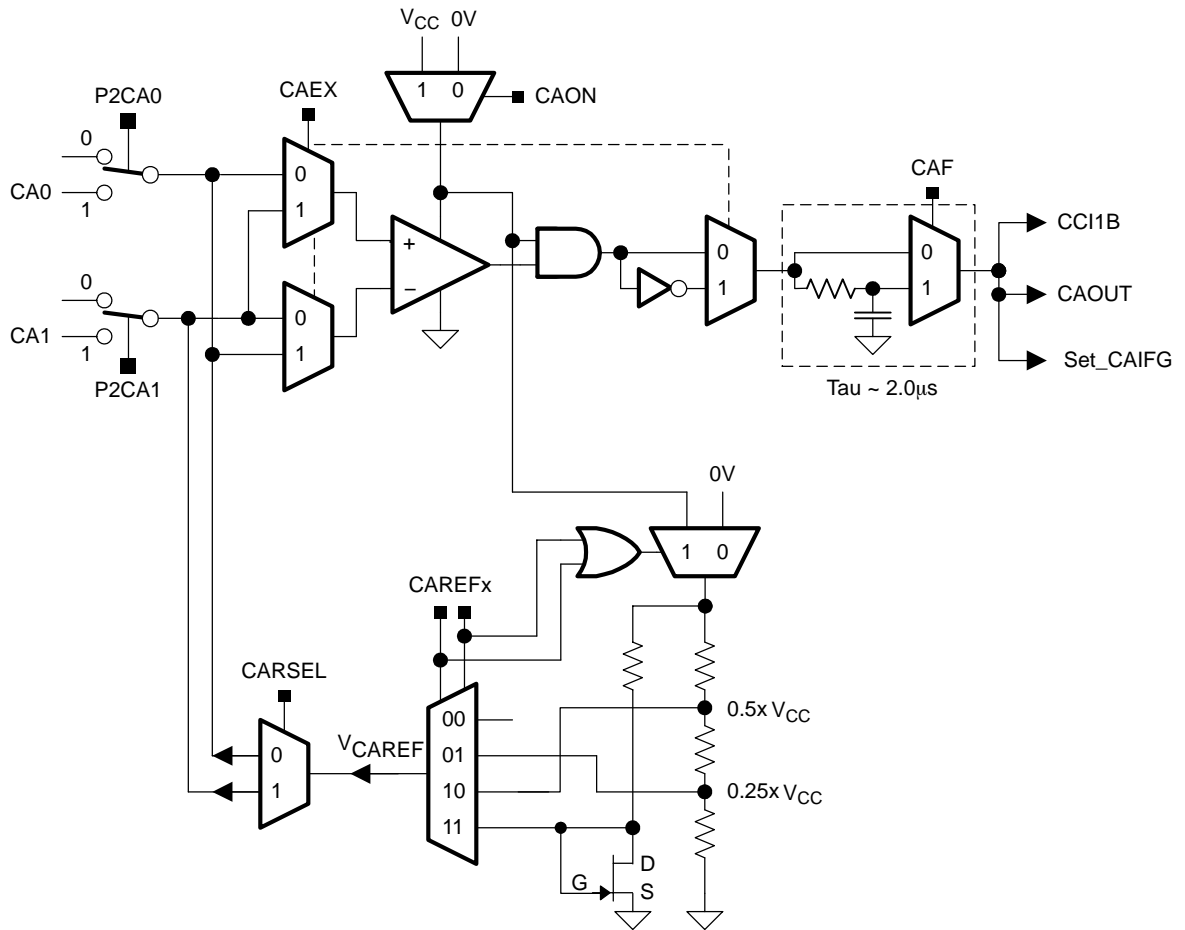
The comparator\_A module supports precision slope analog-to-digital conversions, supply voltage supervision, and monitoring of external analog signals.

Features of Comparator\_A include:

- Inverting and non-inverting terminal input multiplexer
- Software selectable RC-filter for the comparator output
- Output provided to Timer\_A capture input
- Software control of the port input buffer
- Interrupt capability
- Selectable reference voltage generator
- Comparator and reference generator can be powered down

The Comparator\_A block diagram is shown in Figure 17–1.

Figure 17-1. Comparator\_A Block Diagram



## 17.2 Comparator\_A Operation

The comparator\_A module is configured with user software. The setup and operation of comparator\_A is discussed in the following sections.

### 17.2.1 Comparator

The comparator compares the analog voltages at the + and – input terminals. If the + terminal is more positive than the – terminal, the comparator output CAOUT is high. The comparator can be switched on or off using control bit CAON. The comparator should be switched off when not in use to reduce current consumption. When the comparator is switched off, the CAOUT is always low.

### 17.2.2 Input Analog Switches

The analog input switches connect or disconnect the two comparator input terminals to associated port pins using the P2CAx bits. Both comparator terminal inputs can be controlled individually. The P2CAx bits allow:

- Application of an external signal to the + and – terminals of the comparator
- Routing of an internal reference voltage to an associated output port pin

Internally, the input switch is constructed as a T-switch to suppress distortion in the signal path.

---

**Note: Comparator Input Connection**

When the comparator is on, the input terminals should be connected to a signal, power, or ground. Otherwise, floating levels may cause unexpected interrupts and increased current consumption.

---

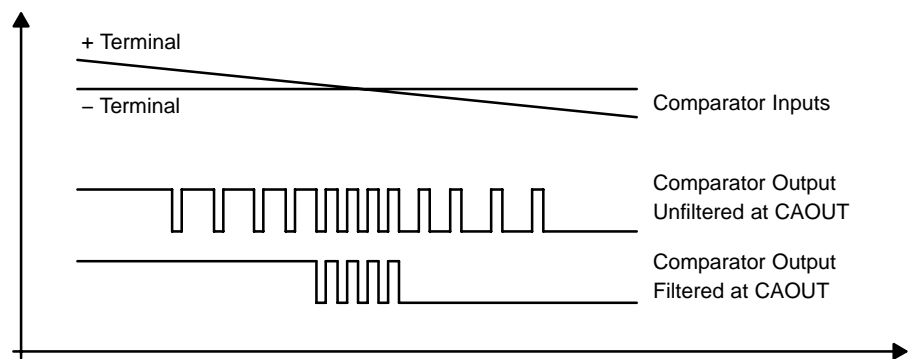
The CAEX bit controls the input multiplexer, exchanging which input signals are connected to the comparator's + and – terminals. Additionally, when the comparator terminals are exchanged, the output signal from the comparator is inverted. This allows the user to determine or compensate for the comparator input offset voltage.

### 17.2.3 Output Filter

The output of the comparator can be used with or without internal filtering. When control bit CAF is set, the output is filtered with an on-chip RC-filter.

Any comparator output oscillates if the voltage difference across the input terminals is small. Internal and external parasitic effects and cross coupling on and between signal lines, power supply lines, and other parts of the system are responsible for this behavior as shown in Figure 17–2. The comparator output oscillation reduces accuracy and resolution of the comparison result. Selecting the output filter can reduce errors associated with comparator oscillation.

Figure 17–2. RC-Filter Response at the Output of the Comparator



### 17.2.4 Voltage Reference Generator

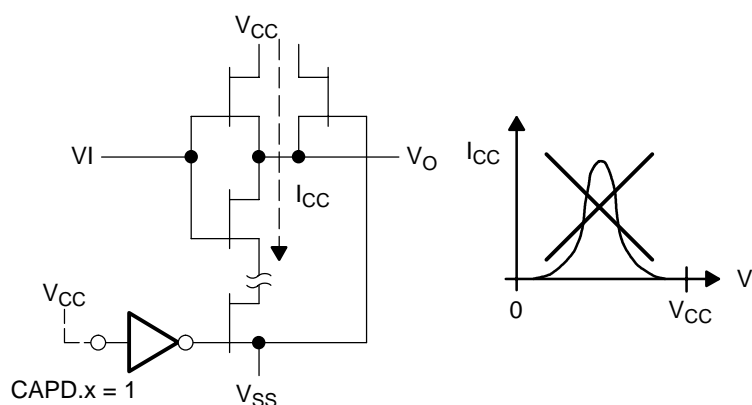
The voltage reference generator is used to generate  $V_{CAREF}$ , which can be applied to either comparator input terminal. The CAREF<sub>x</sub> bits control the output of the voltage generator. The CARSEL bit selects the comparator terminal to which  $V_{CAREF}$  is applied. If external signals are applied to both comparator input terminals, the internal reference generator should be turned off to reduce current consumption. The voltage reference generator can generate a fraction of the device's  $V_{CC}$  or a fixed transistor threshold voltage of  $\sim 0.55$  V.

### 17.2.5 Comparator\_A, Port Disable Register CAPD

The comparator input and output functions are multiplexed with the associated I/O port pins, which are digital CMOS gates. When analog signals are applied to digital CMOS gates, parasitic current can flow from  $V_{CC}$  to GND. This parasitic current occurs if the input voltage is near the transition level of the gate. Disabling the port pin buffer eliminates the parasitic current flow and therefore reduces overall current consumption.

The CAPDx bits, when set, disable the corresponding P1 input buffer as shown in Figure 17–3. When current consumption is critical, any P1 pin connected to analog signals should be disabled with their associated CAPDx bit.

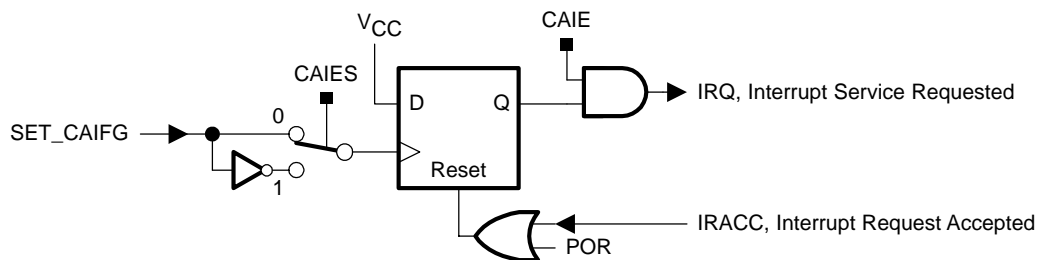
Figure 17–3. Transfer Characteristic and Power Dissipation in a CMOS Inverter/Buffer



### 17.2.6 Comparator\_A Interrupts

One interrupt flag and one interrupt vector are associated with the Comparator\_A as shown in Figure 17–4. The interrupt flag CAIFG is set on either the rising or falling edge of the comparator output, selected by the CAIES bit. If both the CAIE and the GIE bits are set, then the CAIFG flag generates an interrupt request. The CAIFG flag is automatically reset when the interrupt request is serviced or may be reset with software.

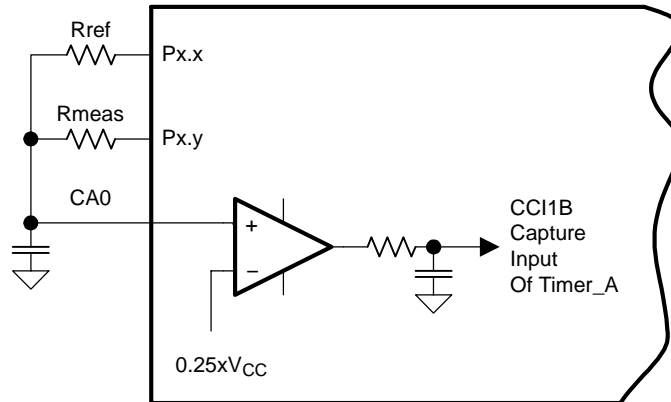
Figure 17–4. Comparator\_A Interrupt System



### 17.2.7 Comparator\_A Used to Measure Resistive Elements

The Comparator\_A can be optimized to precisely measure resistive elements using single slope analog-to-digital conversion. For example, temperature can be converted into digital data using a thermistor, by comparing the thermistor's capacitor discharge time to that of a reference resistor as shown in Figure 17–5. A reference resistor  $R_{ref}$  is compared to  $R_{meas}$ .

Figure 17–5. Temperature Measurement System



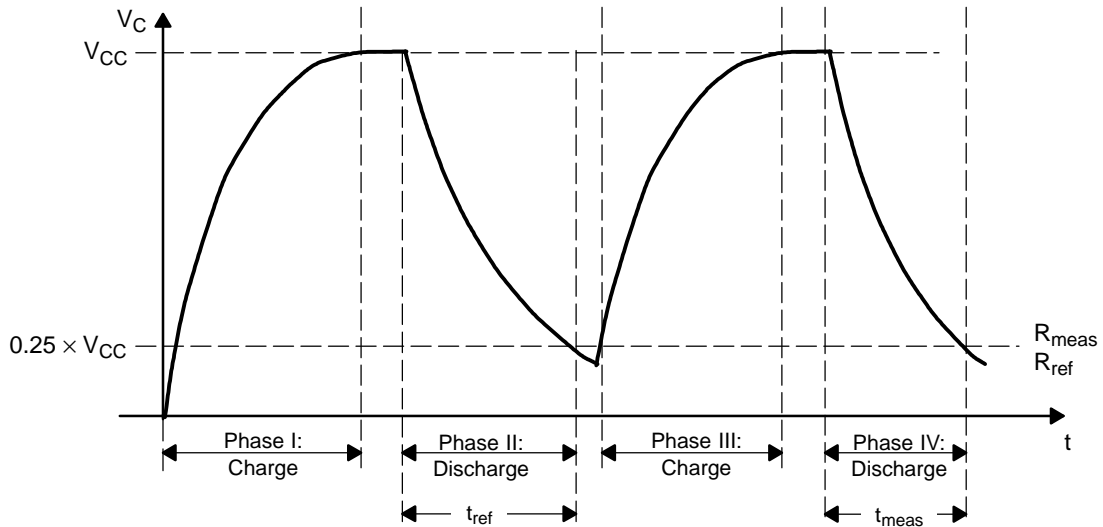
The MSP430 resources used to calculate the temperature sensed by  $R_{meas}$  are:

- Two digital I/O pins to charge and discharge the capacitor.
- I/O set to output high ( $V_{CC}$ ) to charge capacitor, reset to discharge.
- I/O switched to high-impedance input with CAPDx set when not in use.
- One output charges and discharges the capacitor via  $R_{ref}$ .
- One output discharges capacitor via  $R_{meas}$ .
- The + terminal is connected to the positive terminal of the capacitor.
- The – terminal is connected to a reference level, for example  $0.25 \times V_{CC}$ .
- The output filter should be used to minimize switching noise.
- CAOUT used to gate Timer\_A CCI1B, capturing capacitor discharge time.

More than one resistive element can be measured. Additional elements are connected to CA0 with available I/O pins and switched to high impedance when not being measured.

The thermistor measurement is based on a ratiometric conversion principle. The ratio of two capacitor discharge times is calculated as shown in Figure 17–6.

Figure 17–6. Timing for Temperature Measurement Systems



The  $V_{CC}$  voltage and the capacitor value should remain constant during the conversion, but are not critical since they cancel in the ratio:

$$\frac{N_{meas}}{N_{ref}} = \frac{-R_{meas} \times C \times \ln \frac{V_{ref}}{V_{CC}}}{-R_{ref} \times C \times \ln \frac{V_{ref}}{V_{CC}}}$$

$$\frac{N_{meas}}{N_{ref}} = \frac{R_{meas}}{R_{ref}}$$

$$R_{meas} = R_{ref} \times \frac{N_{meas}}{N_{ref}}$$



## 17.3 Comparator\_A Registers

The Comparator\_A registers are listed in Table 17–1.

*Table 17–1. Comparator\_A Registers*

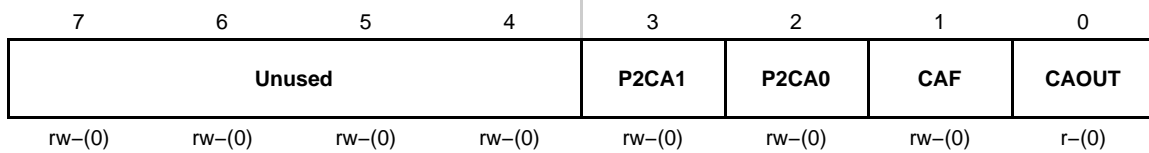
<b>Register</b>	<b>Short Form</b>	<b>Register Type</b>	<b>Address</b>	<b>Initial State</b>
Comparator_A control register 1	CACTL1	Read/write	059h	Reset with POR
Comparator_A control register 2	CACTL2	Read/write	05Ah	Reset with POR
Comparator_A port disable	CAPD	Read/write	05Bh	Reset with POR

**CACTL1, Comparator\_A Control Register 1**

7	6	5	4	3	2	1	0
<b>CAEX</b>	<b>CARSEL</b>	<b>CAREF<sub>x</sub></b>		<b>CAON</b>	<b>CAIES</b>	<b>CAIE</b>	<b>CAIFG</b>
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

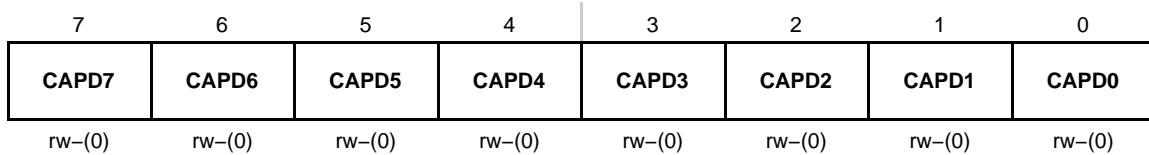
- CAEX**      Bit 7      Comparator\_A exchange. This bit exchanges the comparator inputs and inverts the comparator output.
- CARSEL**   Bit 6      Comparator\_A reference select. This bit selects which terminal the  $V_{CAREF}$  is applied to.  
When CAEX = 0:  
0       $V_{CAREF}$  is applied to the + terminal  
1       $V_{CAREF}$  is applied to the – terminal  
When CAEX = 1:  
0       $V_{CAREF}$  is applied to the – terminal  
1       $V_{CAREF}$  is applied to the + terminal
- CAREF**      Bits 5-4      Comparator\_A reference. These bits select the reference voltage  $V_{CAREF}$ .  
00      Internal reference off. An external reference can be applied.  
01       $0.25 \cdot V_{CC}$   
10       $0.50 \cdot V_{CC}$   
11      Diode reference is selected
- CAON**      Bit 3      Comparator\_A on. This bit turns on the comparator. When the comparator is off it consumes no current. The reference circuitry is enabled or disabled independently.  
0      Off  
1      On
- CAIES**      Bit 2      Comparator\_A interrupt edge select  
0      Rising edge  
1      Falling edge
- CAIE**      Bit 1      Comparator\_A interrupt enable  
0      Disabled  
1      Enabled
- CAIFG**      Bit 0      The Comparator\_A interrupt flag  
0      No interrupt pending  
1      Interrupt pending

**CACTL2, Comparator\_A Control Register 2**



<b>Unused</b>	Bits 7-4	Unused.
<b>P2CA1</b>	Bit 3	Pin to CA1. This bit selects the CA1 pin function. 0 The pin is not connected to CA1 1 The pin is connected to CA1
<b>P2CA0</b>	Bit 2	Pin to CA0. This bit selects the CA0 pin function. 0 The pin is not connected to CA0 1 The pin is connected to CA0
<b>CAF</b>	Bit 1	Comparator_A output filter 0 Comparator_A output is not filtered 1 Comparator_A output is filtered
<b>CAOUT</b>	Bit 0	Comparator_A output. This bit reflects the value of the comparator output. Writing this bit has no effect.

**CAPD, Comparator\_A Port Disable Register**



<b>CAPDx</b>	Bits 7-0	Comparator_A port disable. These bits individually disable the input buffer for the pins of the port associated with Comparator_A. For example, the CAPDx bits can be used to individually enable or disable each P1.x pin buffer. CAPD0 disables P1.0, CAPD1 disables P1.1, etc. 0 The input buffer is enabled. 1 The input buffer is disabled.
--------------	-------------	--

# LCD Controller

---

---

---

---

The LCD controller drives static, 2-mux, 3-mux, or 4-mux LCDs. This chapter describes LCD controller. The LCD controller is implemented on all MSP430x4xx devices, except the MSP430x42x0 devices.

<b>Topic</b>	<b>Page</b>
<b>18.1 LCD Controller Introduction .....</b>	<b>18-2</b>
<b>18.2 LCD Controller Operation .....</b>	<b>18-4</b>
<b>18.3 LCD Controller Registers .....</b>	<b>18-18</b>

## 18.1 LCD Controller Introduction

The LCD controller directly drives LCD displays by creating the ac segment and common voltage signals automatically. The MSP430 LCD controller can support static, 2-mux, 3-mux, and 4-mux LCDs.

The LCD controller features are:

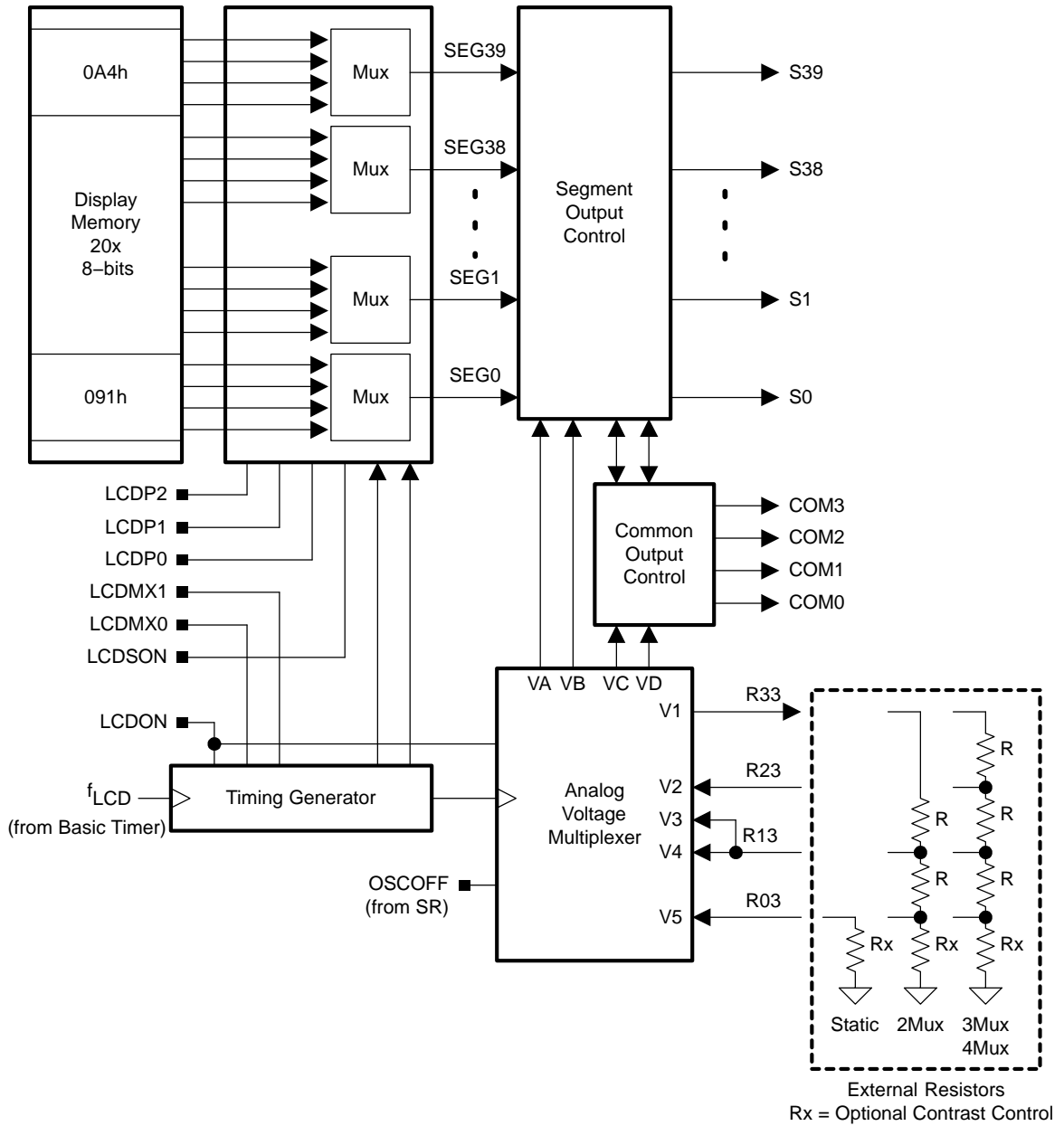
- Display memory
- Automatic signal generation
- Configurable frame frequency
- Blinking capability
- Support for 4 types of LCDs:
  - Static
  - 2-mux, 1/2 bias
  - 3-mux, 1/3 bias
  - 4-mux, 1/3 bias

The LCD controller block diagram is shown in Figure 18–1.

**Note: Max LCD Segment Control**

The maximum number of segment lines available differs with device. See the device-specific datasheet for details.

Figure 18–1. LCD Controller Block Diagram



## 18.2 LCD Controller Operation

The LCD controller is configured with user software. The setup and operation of LCD controller is discussed in the following sections.

### 18.2.1 LCD Memory

The LCD memory map is shown in Figure 18–2. Each memory bit corresponds to one LCD segment, or is not used, depending on the mode. To turn on an LCD segment, its corresponding memory bit is set.

Figure 18–2. LCD memory

Associated Common Pins	3   2   1   0   3   2   1   0								Associated Segment Pins	
	Address								7	0
0A4h	--	--	--	--	--	--	--	--	38	39, 38
0A3h	--	--	--	--	--	--	--	--	36	37, 36
0A2h	--	--	--	--	--	--	--	--	34	35, 34
0A1h	--	--	--	--	--	--	--	--	32	33, 32
0A0h	--	--	--	--	--	--	--	--	30	31, 30
09Fh	--	--	--	--	--	--	--	--	28	29, 28
09Eh	--	--	--	--	--	--	--	--	26	27, 26
09Dh	--	--	--	--	--	--	--	--	24	25, 24
09Ch	--	--	--	--	--	--	--	--	22	23, 22
09Bh	--	--	--	--	--	--	--	--	20	21, 20
09Ah	--	--	--	--	--	--	--	--	18	19, 18
099h	--	--	--	--	--	--	--	--	16	17, 16
098h	--	--	--	--	--	--	--	--	14	15, 14
097h	--	--	--	--	--	--	--	--	12	13, 12
096h	--	--	--	--	--	--	--	--	10	11, 10
095h	--	--	--	--	--	--	--	--	8	9, 8
094h	--	--	--	--	--	--	--	--	6	7, 6
093h	--	--	--	--	--	--	--	--	4	5, 4
092h	--	--	--	--	--	--	--	--	2	3, 2
091h	--	--	--	--	--	--	--	--	0	1, 0

└──────────┘
└──────────┘

$S_{n+1}$ 
 $S_n$

### 18.2.2 Blinking the LCD

The LCD controller supports blinking. The LCDSON bit is ANDed with each segment's memory bit. When LCDSON = 1, each segment is on or off according to its bit value. When LCDSON = 0, each LCD segment is off.

### 18.2.3 LCD Timing Generation

The LCD controller uses the  $f_{LCD}$  signal from the Basic Timer1 to generate the timing for common and segment lines. The proper frequency  $f_{LCD}$  depends on the LCD's requirement for framing frequency and LCD multiplex rate. See the *Basic Timer1* chapter for more information on configuring the  $f_{LCD}$  frequency.

### 18.2.4 LCD Voltage Generation

The voltages required for the LCD signals are supplied externally to pins R33, R23, R13, and R03. Using an equally weighted resistor divider ladder between these pins establishes the analog voltages as shown in Table 18–1. The resistor value R is typically 680 k $\Omega$ . Values of R from 100k $\Omega$  to 1M $\Omega$  can be used depending on LCD requirements.

R33 is a switched- $V_{CC}$  output. This allows the power to the resistor ladder to be turned off eliminating current consumption when the LCD is not used.

Table 18–1. External LCD Module Analog Voltage

OSCOFF	LCDMXx	LCDON	VA	VB	VC	VD	R33
x	xx	0	0	0	0	0	Off
1	xx	x	0	0	0	0	Off
0	00	1	V5/V1	V1/V5	V5/V1	V1/V5	On
0	01	1	V5/V1	V1/V5	V3/V3	V1/V5	On
0	1x	1	V5/V1	V2/V4	V4/V2	V1/V5	On

### LCD Contrast Control

LCD contrast can be controlled by the R03 voltage level with external circuitry, typically an additional resistor Rx to GND. Increasing the voltage at R03 reduces the total applied segment voltage decreasing the LCD contrast.

### 18.2.5 LCD Outputs

Some LCD segment, common, and Rxx functions are multiplexed with digital I/O functions. These pins can function either as digital I/O or as LCD functions. The pin functions for COMx and Rxx, when multiplexed with digital I/O, are selected using the applicable PxSELx bits as described in the *Digital I/O* chapter. The LCD segment functions, when multiplexed with digital I/O, are selected using the LCDPx bits.

The LCDPx bits selects the LCD function for groups of pins. When LCDPx = 0, no multiplexed pin is set to LCD function. When LCDPx = 1, segments S0 - S15 are selected as LCD function. When LCDPx > 1, LCD segment functions are selected in groups of four. For example, when LCDPx = 2, segments S0-S19 are selected as LCD function.

**Note: LCDPx Bits Do Not Affect Dedicated LCD Segment Pins**

The LCDPx bits only affect pins with multiplexed LCD segment functions and digital I/O functions. Dedicated LCD segment pins are not affected by the LCDPx bits.



### 18.2.6 Static Mode

In static mode, each MSP430 segment pin drives one LCD segment and one common line, COM0, is used. Figure 18–3 shows some example static waveforms.

Figure 18–3. Example Static Waveforms

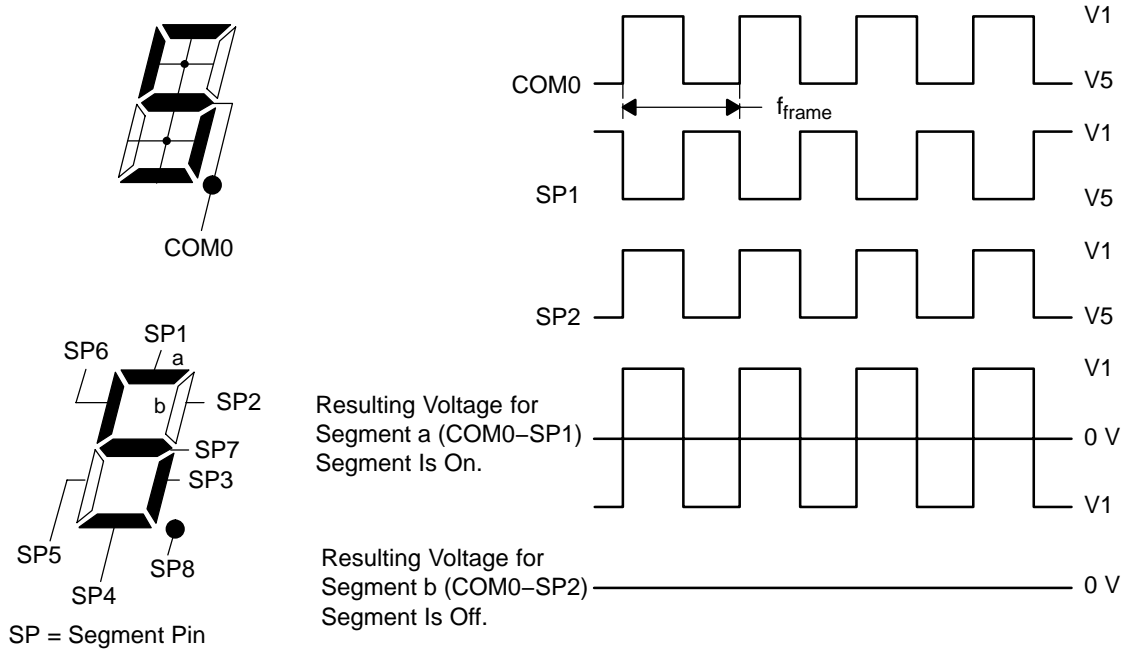
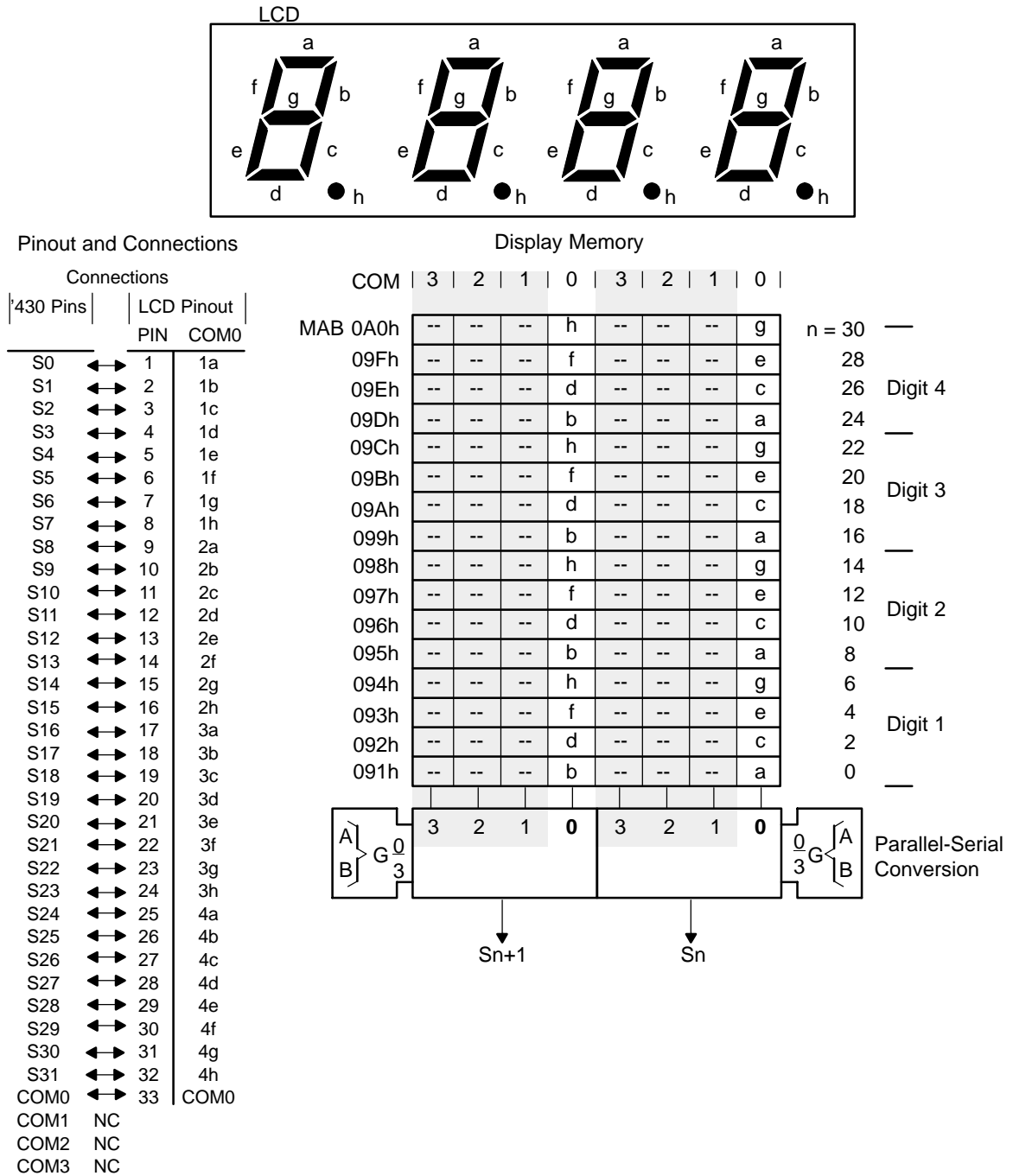


Figure 18–4 shows an example static LCD, pin-out, LCD-to-MSP430 connections, and the resulting segment mapping. This is only an example. Segment mapping in a user’s application depends on the LCD pin-out and on the MSP430-to-LCD connections.

Figure 18–4. Static LCD Example



## Static Mode Software Example

```

; All eight segments of a digit are often located in four
; display memory bytes with the static display method.
;
a    EQU    001h
b    EQU    010h
c    EQU    002h
d    EQU    020h
e    EQU    004h
f    EQU    040h
g    EQU    008h
h    EQU    080h
; The register content of Rx should be displayed.
; The Table represents the 'on'-segments according to the
; content of Rx.
    MOV.B Table (Rx),RY    ; Load segment information
                           ; into temporary memory.
                           ; (Ry) = 0000 0000 hfdb geca
    MOV.B Ry,&LCDn         ; Note:
                           ; All bits of an LCD memory
                           ; ' byte are written
                           ; (Ry) = 0000 0000 0hfd bgec
    RRA    Ry
    MOV.B Ry,&LCDn+1       ; Note:
                           ; All bits of an LCD memory
                           ; byte are written
                           ; (Ry) = 0000 0000 00hf dbge
    RRA    Ry
    MOV.B Ry,&LCDn+2       ; Note:
                           ; All bits of an LCD memory
                           ; ' byte are written
                           ; (Ry) = 0000 0000 000h fdbg
    RRA    Ry
    MOV.B Ry,&LCDn+3       ; Note:
                           ; All bits of an LCD memory
                           ; ' byte are written

    .....
    .....
;
Table DB    a+b+c+d+e+f    ; displays "0"
      DB    b+c;          ; displays "1"
      .....
      .....
      DB
      .....

```

### 18.2.7 2-Mux Mode

In 2-mux mode, each MSP430 segment pin drives two LCD segments and two common lines, COM0 and COM1, are used. Figure 18–5 shows some example 2-mux waveforms.

Figure 18–5. Example 2-Mux Waveforms

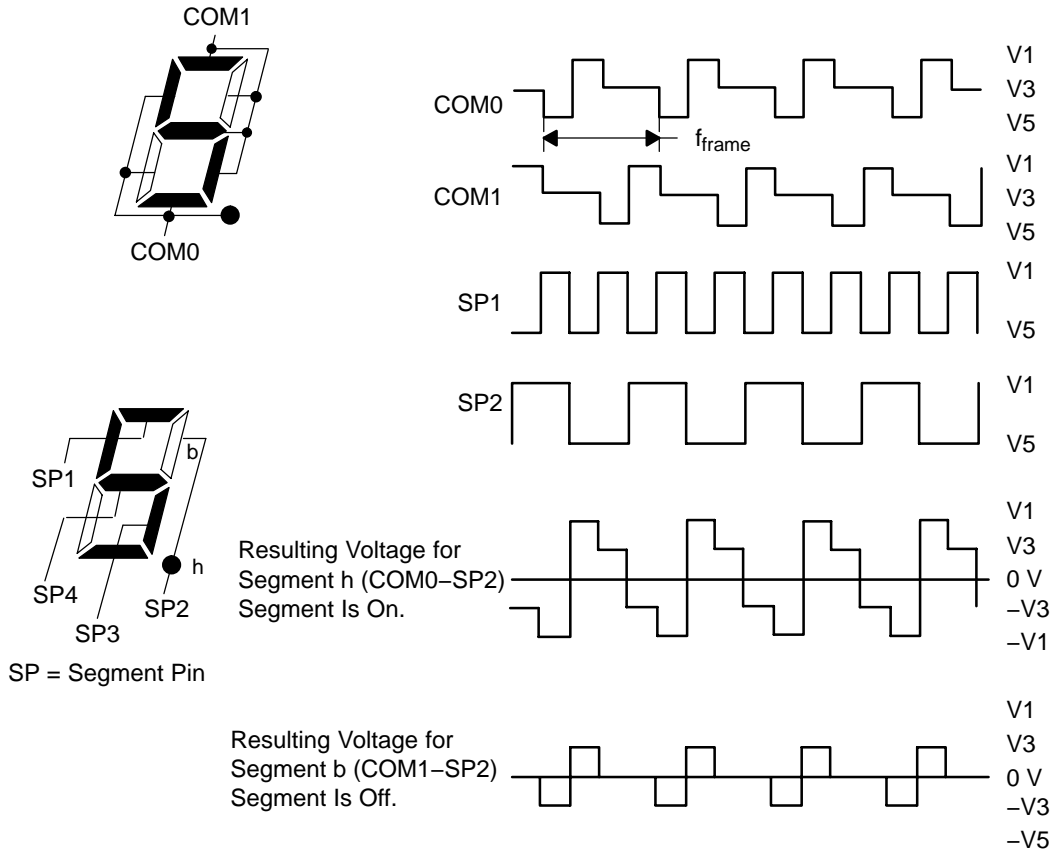
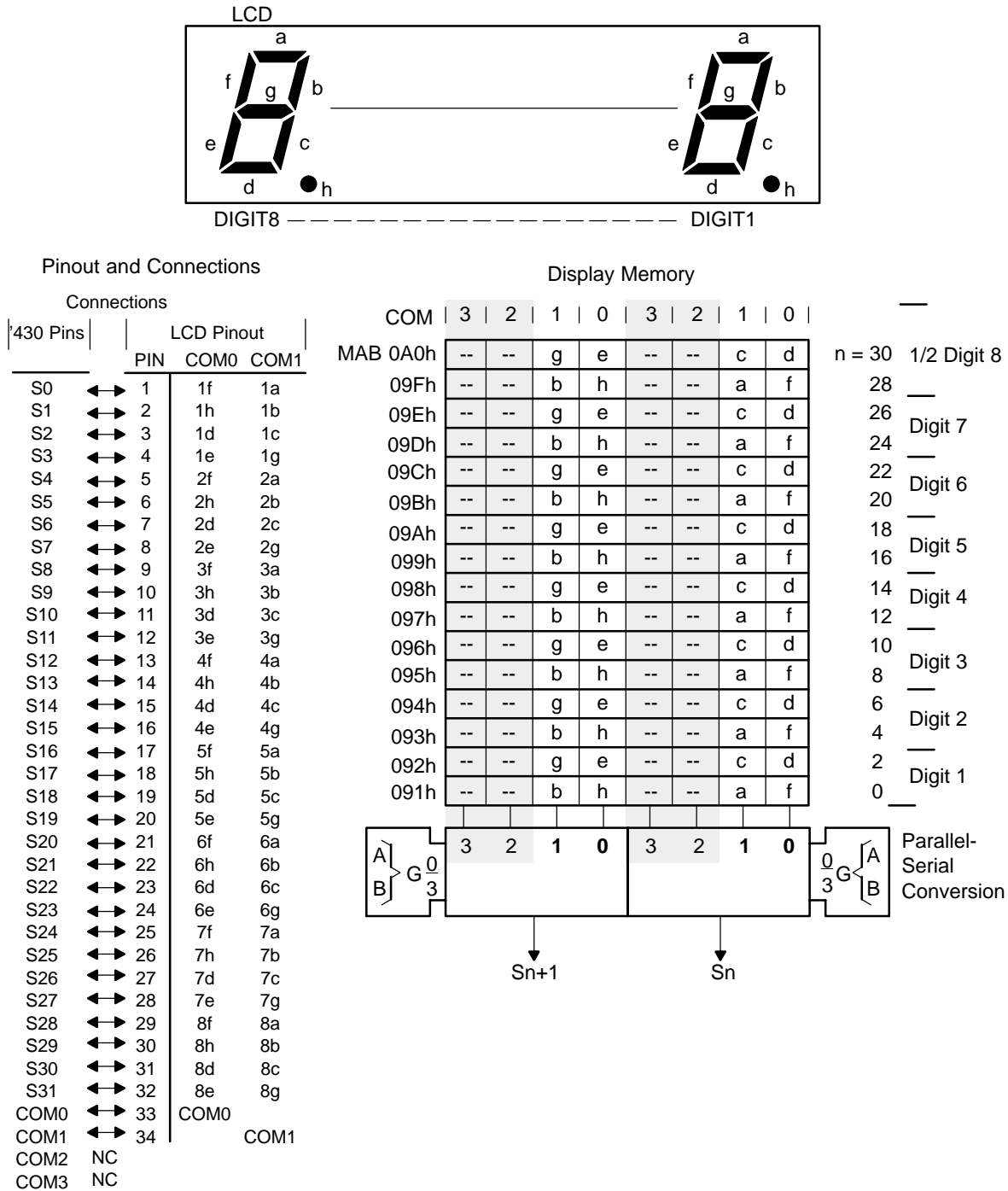


Figure 18–6 shows an example 2-mux LCD, pin-out, LCD-to-MSP430 connections, and the resulting segment mapping. This is only an example. Segment mapping in a user’s application completely depends on the LCD pin-out and on the MSP430-to-LCD connections.

Figure 18–6. 2–Mux LCD Example



## 2-Mux Mode Software Example

```

; All eight segments of a digit are often located in two
; display memory bytes with the 2mux display rate
;
a EQU 002h
b EQU 020h
c EQU 008h
d EQU 004h
e EQU 040h
f EQU 001h
g EQU 080h
h EQU 010h
; The register content of Rx should be displayed.
; The Table represents the 'on'-segments according to the
; content of Rx.
;
.....
.....
MOV.B Table(Rx),Ry ; Load segment information into
; temporary memory.
MOV.B Ry,&LCDn ; (Ry) = 0000 0000 gebh cdaf
; Note:
; All bits of an LCD memory byte
; are written
RRA Ry ; (Ry) = 0000 0000 0geb hcda
RRA Ry ; (Ry) = 0000 0000 00ge bhcd
MOV.B Ry,&LCDn+1 ; Note:
; All bits of an LCD memory byte
; are written
.....
.....
.....
Table DB a+b+c+d+e+f ; displays "0"
.....
DB a+b+c+d+e+f+g+ ; displays "8"
.....
.....
DB
.....
;

```

### 18.2.8 3-Mux Mode

In 3-mux mode, each MSP430 segment pin drives three LCD segments and three common lines, COM0, COM1 and COM2 are used. Figure 18–7 shows some example 3-mux waveforms.

Figure 18–7. Example 3-Mux Waveforms

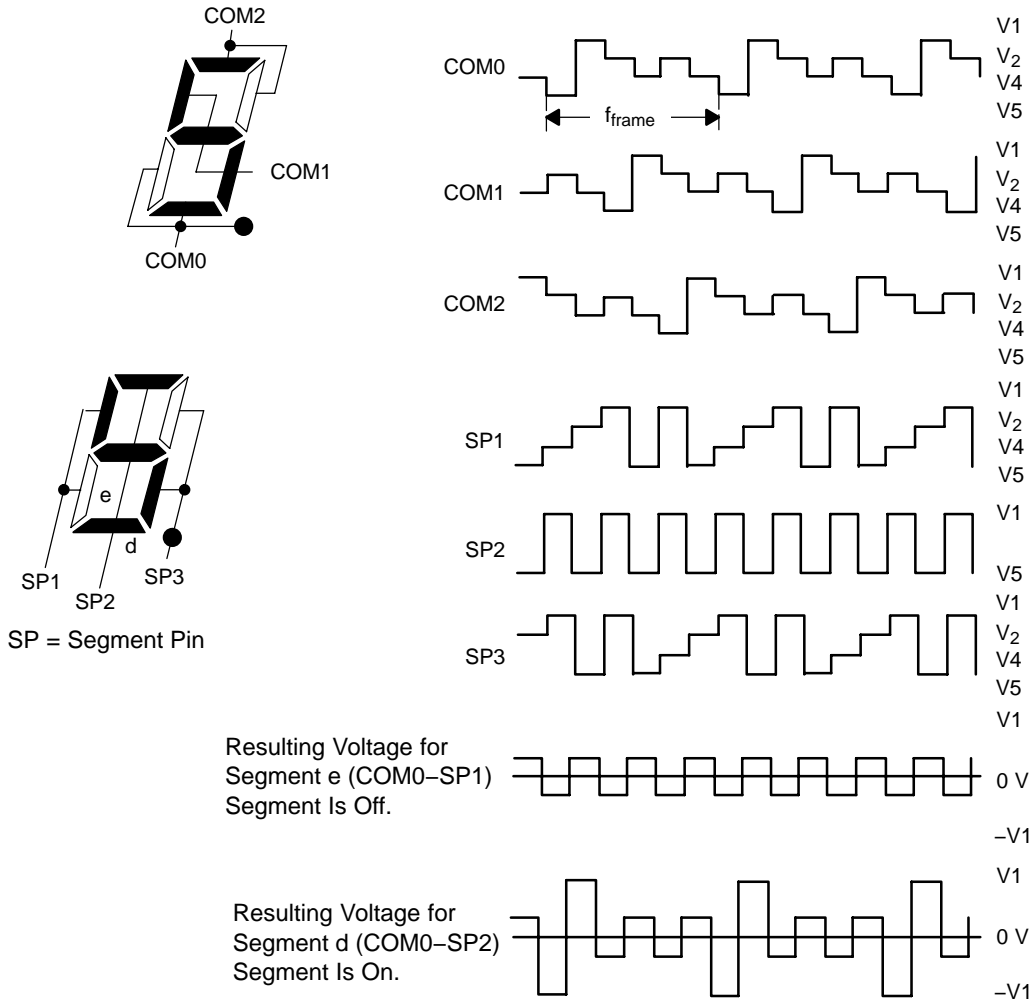
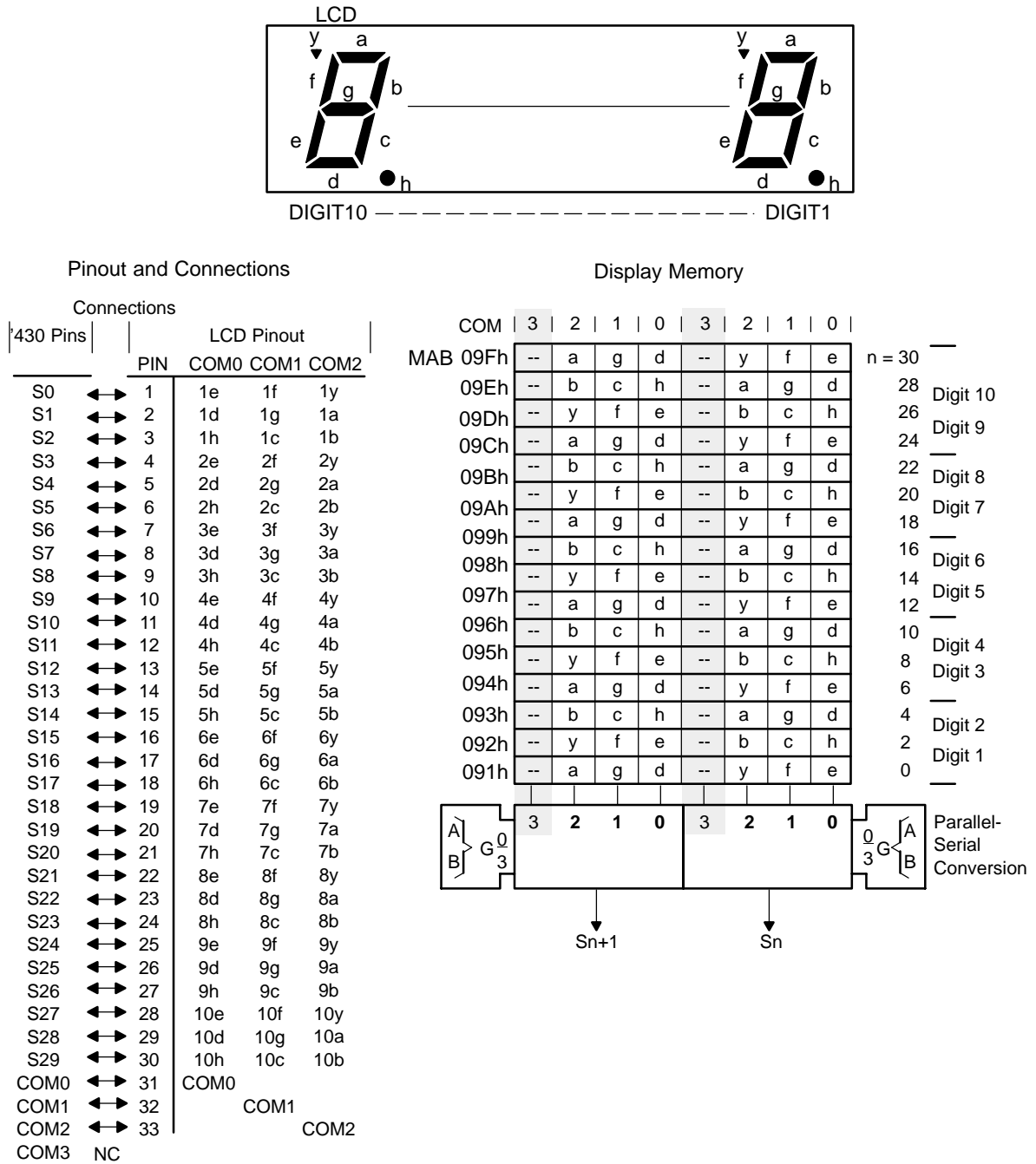


Figure 18–8 shows an example 3-mux LCD, pin-out, LCD-to-MSP430 connections, and the resulting segment mapping. This is only an example. Segment mapping in a user’s application depends on the LCD pin-out and on the MSP430-to-LCD connections.

Figure 18–8. 3-Mux LCD Example





## 3-Mux Mode Software Example

```

; The 3mux rate can support nine segments for each
; digit. The nine segments of a digit are located in
; 1 1/2 display memory bytes.
;
a EQU 0040h
b EQU 0400h
c EQU 0200h
d EQU 0010h
e EQU 0001h
f EQU 0002h
g EQU 0020h
h EQU 0100h
Y EQU 0004h
; The LSDigit of register Rx should be displayed.
; The Table represents the 'on'-segments according to the
; LSDigit of register of Rx.
; The register Ry is used for temporary memory
;
ODDDIGRLA Rx ; LCD in 3mux has 9 segments per
; digit; word table required for
; displayed characters.
MOV Table(Rx),Ry ; Load segment information to
; temporary mem.
; (Ry) = 0000 0bch 0agd 0yfe
MOV.B Ry,&LCDn ; write 'a, g, d, y, f, e' of
; Digit n (LowByte)
SWPB Ry ; (Ry) = 0agd 0yfe 0000 0bch
BIC.B #07h,&LCDn+1 ; write 'b, c, h' of Digit n
; (HighByte)
BIS.B Ry,&LCDn+1
.....
EVNDIGRLA Rx ; LCD in 3mux has 9 segments per
; digit; word table required for
; displayed characters.
MOV Table(Rx),Ry ; Load segment information to
; temporary mem.
; (Ry) = 0000 0bch 0agd 0yfe
RLA Ry ; (Ry) = 0000 bch0 agd0 yfe0
RLA Ry ; (Ry) = 000b ch0a gd0y fe00
RLA Ry ; (Ry) = 00bc h0ag d0yf e000
RLA Ry ; (Ry) = 0bch 0agd 0yfe 0000
BIC.B #070h,&LCDn+1
BIS.B Ry,&LCDn+1 ; write 'y, f, e' of Digit n+1
; (LowByte)
SWPB Ry ; (Ry) = 0yfe 0000 0bch 0agd
MOV.B Ry,&LCDn+2 ; write 'b, c, h, a, g, d' of
; Digit n+1 (HighByte)
.....
Table DW a+b+c+d+e+f ; displays "0"
DW b+c ; displays "1"
.....
DW a+e+f+g ; displays "F"

```

### 18.2.9 4-Mux Mode

In 4-mux mode, each MSP430 segment pin drives four LCD segments and all four common lines, COM0, COM1, COM2, and COM3 are used. Figure 18–9 shows some example 4-mux waveforms.

Figure 18–9. Example 4-Mux Waveforms

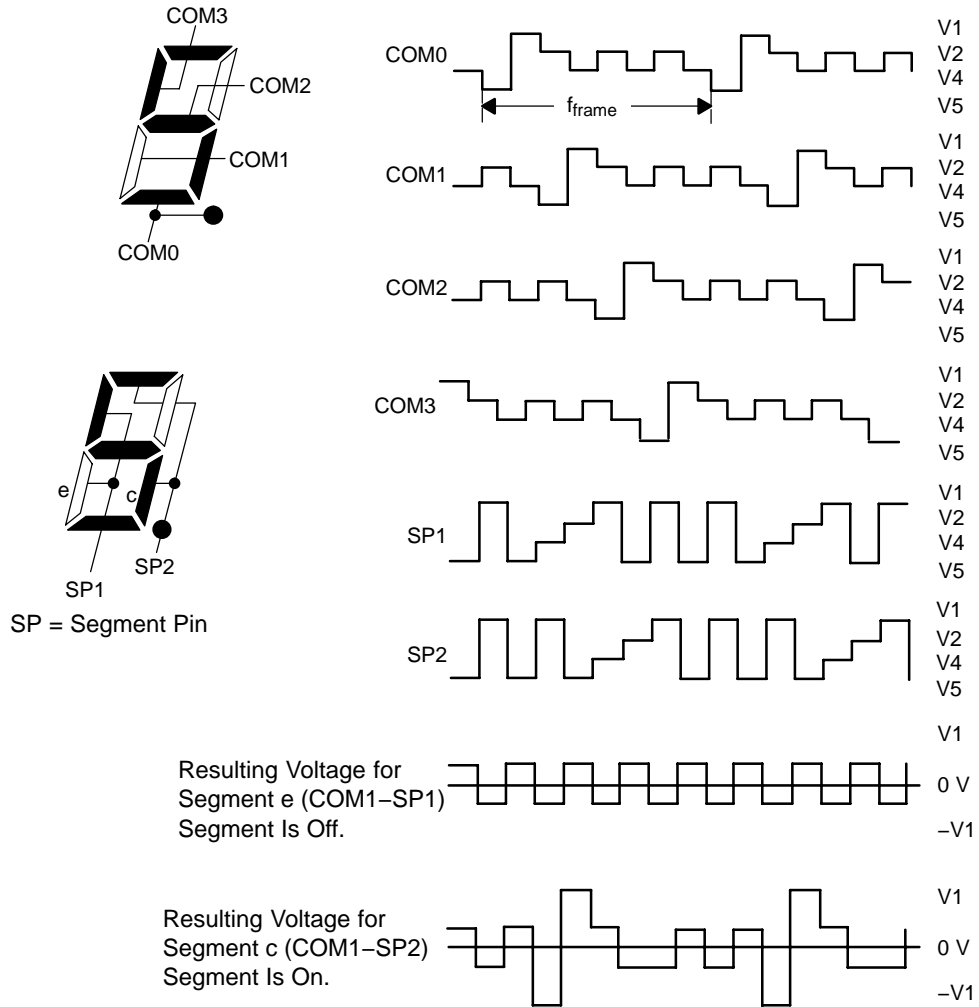
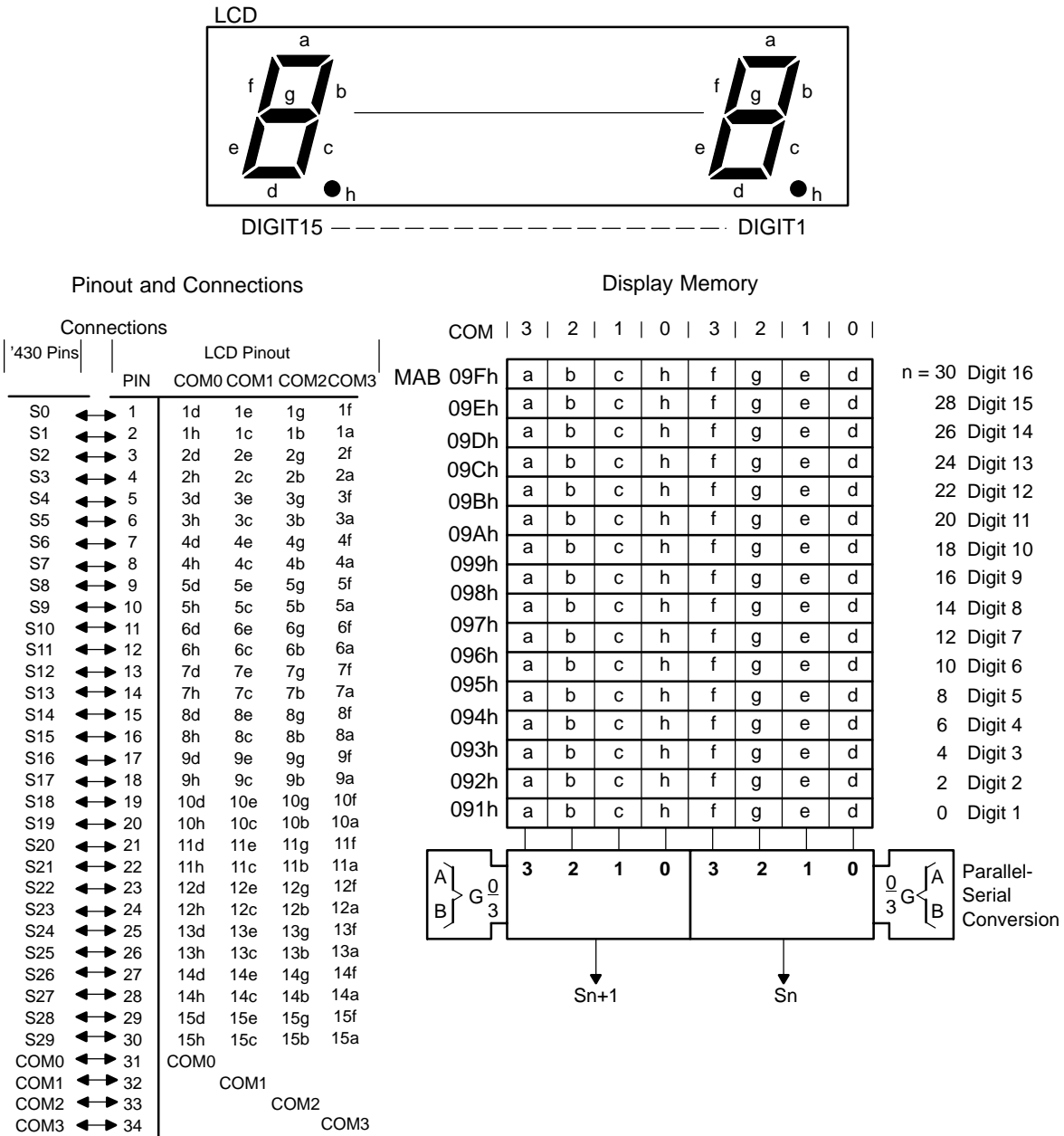


Figure 18–10 shows an example 4-mux LCD, pin-out, LCD-to-MSP430 connections, and the resulting segment mapping. This is only an example. Segment mapping in a user’s application depends on the LCD pin-out and on the MSP430-to-LCD connections.

Figure 18–10. 4-Mux LCD Example



## 4-Mux Mode Software Example

```

; The 4mux rate supports eight segments for each digit.
; All eight segments of a digit can often be located in
; one display memory byte
a    EQU    080h
b    EQU    040h
c    EQU    020h
d    EQU    001h
e    EQU    002h
f    EQU    008h
g    EQU    004h
h    EQU    010h
;
; The LSDigit of register Rx should be displayed.
; The Table represents the 'on'-segments according to the
; content of Rx.
;
    MOV.B Table(Rx),&LCDn ; n = 1 ..... 15
                           ; all eight segments are
                           ; written to the display
                           ; memory
    .....
    .....
Table DB    a+b+c+d+e+f    ; displays "0"
      DB    b+c            ; displays "1"
    .....
    .....
      DB    b+c+d+e+g      ; displays "d"
      DB    a+d+e+f+g      ; displays "E"
      DB    a+e+f+g        ; displays "F"

```

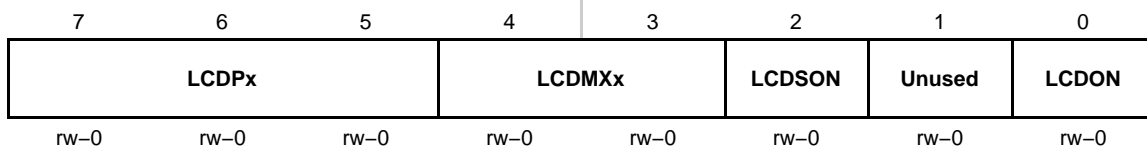
### 18.3 LCD Controller Registers

The LCD Controller registers are listed in Table 18–2.

Table 18–2. LCD Controller Registers

Register	Short Form	Register Type	Address	Initial State
LCD control register	LCDCTL	Read/write	090h	Reset with PUC
LCD memory 1	LCDM1	Read/write	091h	Unchanged
LCD memory 2	LCDM2	Read/write	092h	Unchanged
LCD memory 3	LCDM3	Read/write	093h	Unchanged
LCD memory 4	LCDM4	Read/write	094h	Unchanged
LCD memory 5	LCDM5	Read/write	095h	Unchanged
LCD memory 6	LCDM6	Read/write	096h	Unchanged
LCD memory 7	LCDM7	Read/write	097h	Unchanged
LCD memory 8	LCDM8	Read/write	098h	Unchanged
LCD memory 9	LCDM9	Read/write	099h	Unchanged
LCD memory 10	LCDM10	Read/write	09Ah	Unchanged
LCD memory 11	LCDM11	Read/write	09Bh	Unchanged
LCD memory 12	LCDM12	Read/write	09Ch	Unchanged
LCD memory 13	LCDM13	Read/write	09Dh	Unchanged
LCD memory 14	LCDM14	Read/write	09Eh	Unchanged
LCD memory 15	LCDM15	Read/write	09Fh	Unchanged
LCD memory 16	LCDM16	Read/write	0A0h	Unchanged
LCD memory 17	LCDM17	Read/write	0A1h	Unchanged
LCD memory 18	LCDM18	Read/write	0A2h	Unchanged
LCD memory 19	LCDM19	Read/write	0A3h	Unchanged
LCD memory 20	LCDM20	Read/write	0A4h	Unchanged

## LCDCTL, LCD Control Register



<b>LCDPx</b>	Bits 7-5	<p>LCD Port Select. These bits select the pin function to be port I/O or LCD function for groups of segments pins. These bits <b>ONLY</b> affect pins with multiplexed functions. Dedicated LCD pins are always LCD function.</p> <p>000 No multiplexed pins are LCD function  001 S0-S15 are LCD function  010 S0-S19 are LCD function  011 S0-S23 are LCD function  100 S0-S27 are LCD function  101 S0-S31 are LCD function  110 S0-S35 are LCD function  111 S0-S39 are LCD function</p>
<b>LCDMXx</b>	Bits 4-3	<p>LCD mux rate. These bits select the LCD mode.</p> <p>00 Static  01 2-mux  10 3-mux  11 4-mux</p>
<b>LCDSON</b>	Bit 2	<p>LCD segments on. This bit supports flashing LCD applications by turning off all segment lines, while leaving the LCD timing generator and R33 enabled.</p> <p>0 All LCD segments are off  1 All LCD segments are enabled and on or off according to their corresponding memory location.</p>
<b>Unused</b>	Bit 1	Unused
<b>LCDON</b>	Bit 0	<p>LCD On. This bit turns on the LCD timing generator and R33.</p> <p>0 LCD timing generator and Ron are off  1 LCD timing generator and Ron are on</p>

# LCD\_A Controller

---

---

---

---

The LCD\_A controller drives static, 2-mux, 3-mux, or 4-mux LCDs. This chapter describes the LCD\_A controller. The LCD\_A controller is implemented on the MSP430x42x0 and MSP430F46xx devices.

<b>Topic</b>	<b>Page</b>
<b>19.1 LCD Controller Introduction</b> .....	<b>19-2</b>
<b>19.2 LCD Controller Operation</b> .....	<b>19-4</b>
<b>19.3 LCD Controller Registers</b> .....	<b>19-21</b>

## 19.1 LCD\_A Controller Introduction

The LCD\_A controller directly drives LCD displays by creating the ac segment and common voltage signals automatically. The MSP430 LCD controller can support static, 2-mux, 3-mux, and 4-mux LCDs.

The LCD controller features are:

- Display memory
- Automatic signal generation
- Configurable frame frequency
- Blinking capability
- Regulated charge pump
- Contrast control by software
- Support for 4 types of LCDs:
  - Static
  - 2-mux, 1/2 bias or 1/3 bias
  - 3-mux, 1/2 bias or 1/3 bias
  - 4-mux, 1/2 bias or 1/3 bias

The LCD controller block diagram is shown in Figure 19–1.

---

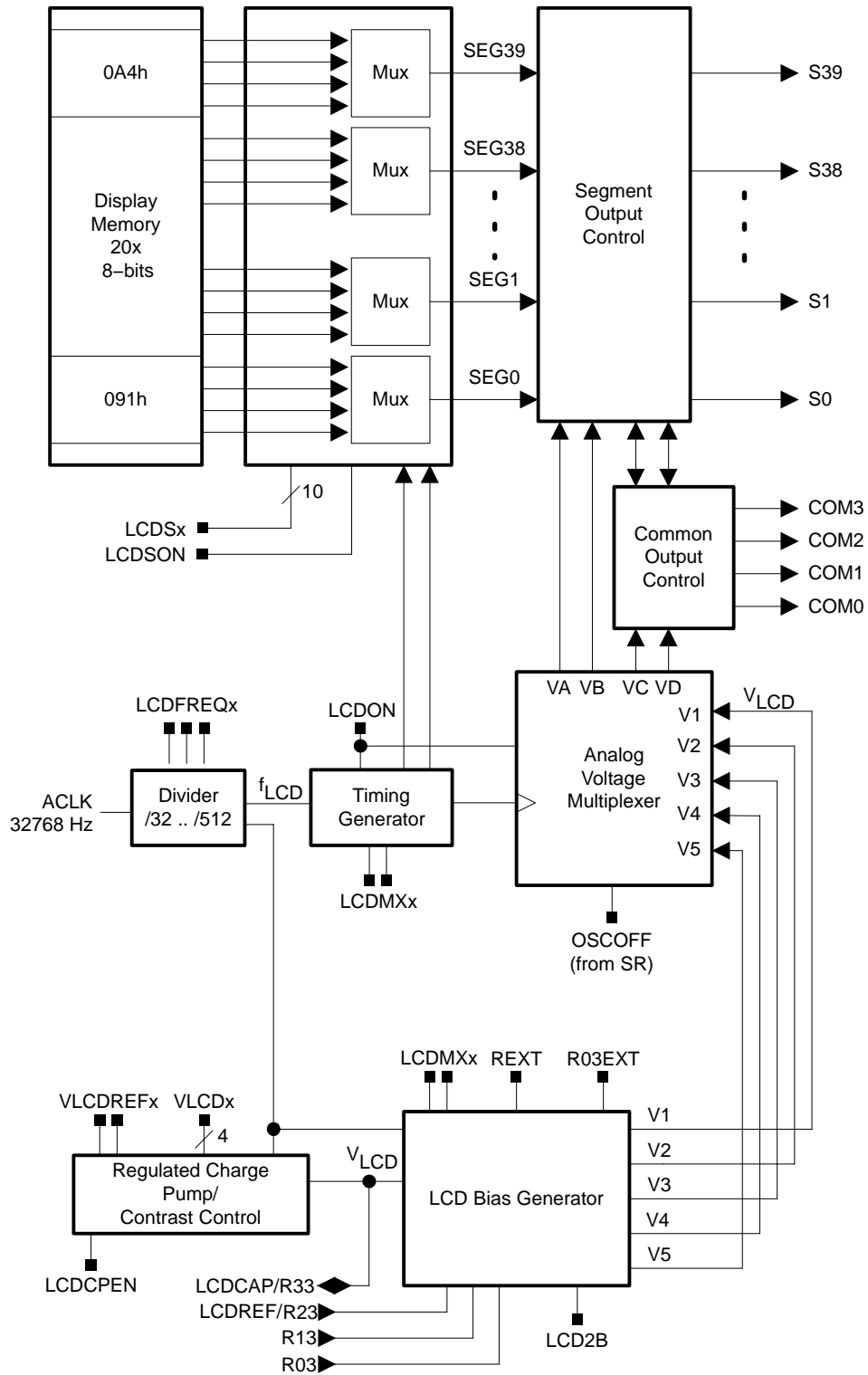
**Note: Maximum LCD Segment Control**

The maximum number of segment lines available differs with device. See the device-specific datasheet for available segment pins.

---



Figure 19–1. LCD\_A Controller Block Diagram



## 19.2 LCD\_A Controller Operation

The LCD\_A controller is configured with user software. The setup and operation of the LCD\_A controller is discussed in the following sections.

### 19.2.1 LCD Memory

The LCD memory map is shown in Figure 19–2. Each memory bit corresponds to one LCD segment, or is not used, depending on the mode. To turn on an LCD segment, its corresponding memory bit is set.

Figure 19–2. LCD memory

Associated Common Pins	3	2	1	0	3	2	1	0	Address	7	0	n	Associated Segment Pins
	--	--	--	--	--	--	--	--	0A4h	--	--	--	38 39, 38
	--	--	--	--	--	--	--	--	0A3h	--	--	--	36 37, 36
	--	--	--	--	--	--	--	--	0A2h	--	--	--	34 35, 34
	--	--	--	--	--	--	--	--	0A1h	--	--	--	32 33, 32
	--	--	--	--	--	--	--	--	0A0h	--	--	--	30 31, 30
	--	--	--	--	--	--	--	--	09Fh	--	--	--	28 29, 28
	--	--	--	--	--	--	--	--	09Eh	--	--	--	26 27, 26
	--	--	--	--	--	--	--	--	09Dh	--	--	--	24 25, 24
	--	--	--	--	--	--	--	--	09Ch	--	--	--	22 23, 22
	--	--	--	--	--	--	--	--	09Bh	--	--	--	20 21, 20
	--	--	--	--	--	--	--	--	09Ah	--	--	--	18 19, 18
	--	--	--	--	--	--	--	--	099h	--	--	--	16 17, 16
	--	--	--	--	--	--	--	--	098h	--	--	--	14 15, 14
	--	--	--	--	--	--	--	--	097h	--	--	--	12 13, 12
	--	--	--	--	--	--	--	--	096h	--	--	--	10 11, 10
	--	--	--	--	--	--	--	--	095h	--	--	--	8 9, 8
	--	--	--	--	--	--	--	--	094h	--	--	--	6 7, 6
	--	--	--	--	--	--	--	--	093h	--	--	--	4 5, 4
	--	--	--	--	--	--	--	--	092h	--	--	--	2 3, 2
	--	--	--	--	--	--	--	--	091h	--	--	--	0 1, 0

⏟
⏟  
 Sn+1                      Sn

### 19.2.2 Blinking the LCD

The LCD controller supports blinking. The LCDSON bit is ANDed with each segment's memory bit. When LCDSON = 1, each segment is on or off according to its bit value. When LCDSON = 0, each LCD segment is off.

### 19.2.3 LCD\_A Voltage And Bias Generation

The LCD\_A module allows selectable sources for the peak output waveform voltage,  $V_1$ , as well as the fractional LCD biasing voltages  $V_2 - V_5$ .  $V_{LCD}$  may be sourced from  $AV_{CC}$ , an internal charge pump, or externally.

All internal voltage generation is disabled if the oscillator sourcing ACLK is turned off (OSCOFF = 1) or the LCD\_A module is disabled (LCDON = 0).

#### LCD Voltage Selection

$V_{LCD}$  is sourced from  $AV_{CC}$  when  $VLCDEXT = 0$ ,  $VLCDx = 0$ , and  $VREFx = 0$ .  $V_{LCD}$  is sourced from the internal charge pump when  $VLCDEXT = 0$ ,  $VLCDPEN = 1$ , and  $VLCDx > 0$ . The charge pump is always sourced from  $DV_{CC}$ . The  $VLCDx$  bits provide a software selectable LCD voltage from 2.6 V to 3.44 V (typical) independent of  $DV_{CC}$ . See the device-specific datasheet for specifications.

When the internal charge pump is used, a 4.7  $\mu\text{F}$  or larger capacitor must be connected between pin LCDCAP and ground. Otherwise, irreversible damage can occur. When the charge pump is enabled, peak currents of 2 mA typical occur on  $DV_{CC}$ . However, the charge pump duty cycle is approximately 1/1000, resulting in a 2  $\mu\text{A}$  average current. The charge pump may be temporarily disabled by setting  $LCDCPEN = 0$  with  $VLCDx > 0$  to reduce system noise. In this case, the voltage present at the external capacitor is used for the LCD voltages until the charge pump is re-enabled.

**Note: Capacitor Required For Internal Charge Pump**

A 4.7  $\mu\text{F}$  or larger capacitor must be connected from pin LCDCAP to ground when the internal charge pump is enabled. Otherwise, damage can occur.

The internal charge pump may use an external reference voltage when  $VLCDREFx = 01$ . In this case, the charge pump voltage will be 3x the voltage applied externally to the LCDREF pin and the  $VLCDx$  bits are ignored.

When  $VLCDEXT = 1$ ,  $V_{LCD}$  is sourced externally from the LCDCAP pin and the internal charge pump is disabled. The charge pump and internal bias generation require an input clock source of 32768 Hz  $\pm 10\%$ . If neither is used, the input clock frequency may be different per the application needs.

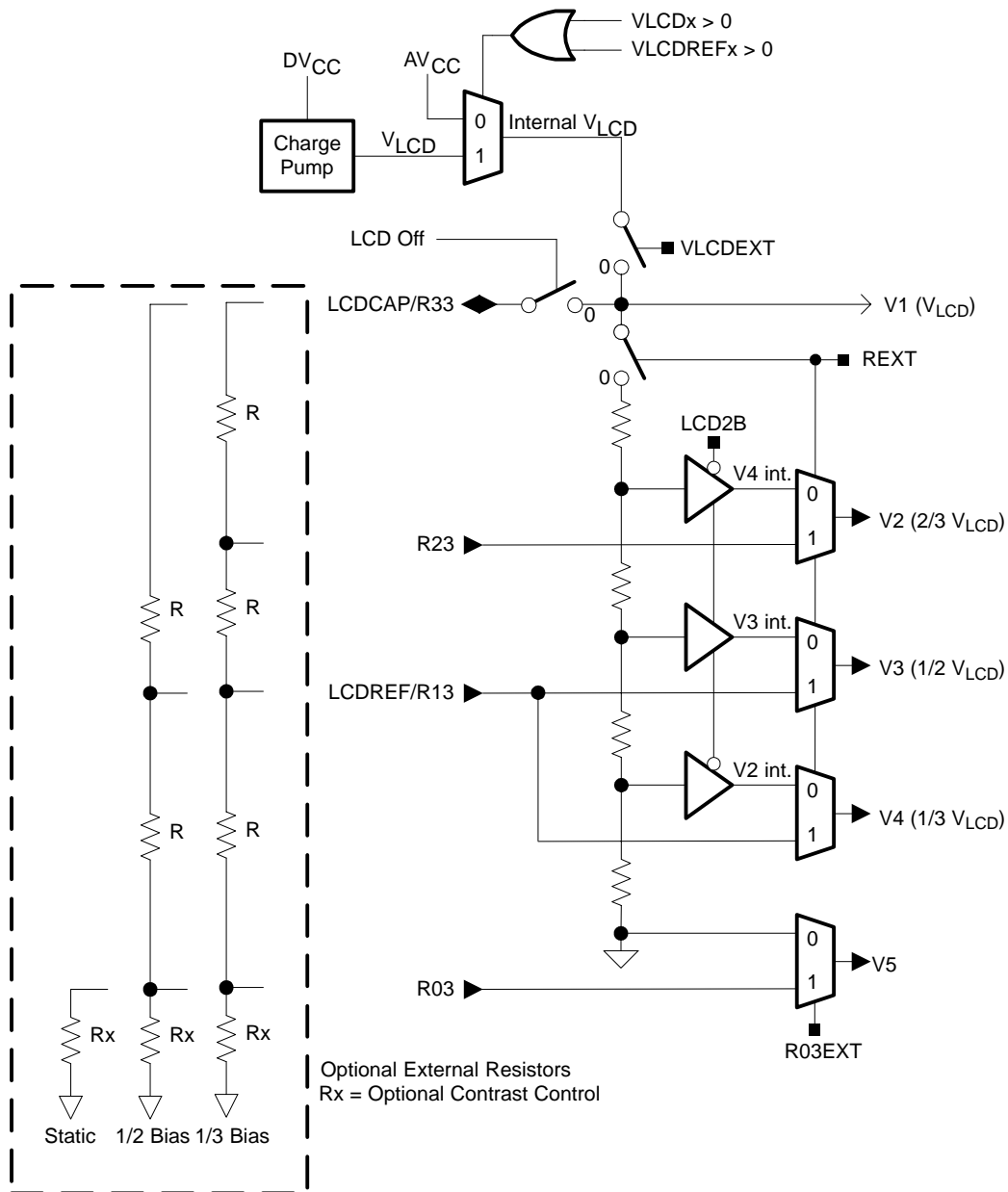
#### LCD Bias Generation

The fractional LCD biasing voltages,  $V_2 - V_5$  can be generated internally or externally, independent of the source for  $V_{LCD}$ . The LCD bias generation block diagram is shown in Figure 19–3.

To source the bias voltages V2 – V4 externally, REXT is set. This also disables the internal bias generation. Typically an equally weighted resistor divider is used with resistors ranging from 100 kΩ to 1 MΩ. When using an external resistor divider, the V<sub>LCD</sub> voltage may be sourced from the internal charge pump when VLCDEXT = 0. V5 can also be sourced externally when R03EXT is set.

When using an external resistor divider R33 may serve as a switched-V<sub>LCD</sub> output when VLCDEXT = 0. This allows the power to the resistor ladder to be turned off eliminating current consumption when the LCD is not used. When VLCDEXT = 1, R33 serves as a V<sub>LCD</sub> input.

Figure 19–3. Bias Generation



The internal bias generator supports 1/2 bias LCDs when LCD2B = 1, and 1/3 bias LCDs when LCD2B = 0 in 2-mux, 3-mux, and 4-mux modes. In static mode the internal divider is disabled.

Some devices share the LCDCAP, R33, and R23 functions. In this case, the charge pump cannot be used together with an external resistor divider with 1/3 biasing. When R03 is not available externally. The V5 is always  $AV_{SS}$ .

## LCD Contrast Control

The peak voltage of the output waveforms together with the selected mode and biasing determine the contrast and the contrast ratio of the LCD. The LCD contrast can be controlled in software by adjusting the LCD voltage generated by the integrated charge pump using the VLCDx settings.

The contrast ratio depends on the used LCD display and the selected biasing scheme. Table 19–1 shows the biasing configurations that apply to the different modes together with the RMS voltages for the segments turned on ( $V_{RMS,ON}$ ) and turned off ( $V_{RMS,OFF}$ ) as functions of  $V_{LCD}$ . It also shows the resulting contrast ratios between the on and off states.

Table 19–1. LCD Voltage and Biasing Characteristics

Mode	Bias Config.	LCDMx	LCD2B	COM Lines	Voltage Levels	$V_{RMS,OFF}/V_{LCD}$	$V_{RMS,ON}/V_{LCD}$	Contrast Ratio $V_{RMS,ON}/V_{RMS,OFF}$
static	static	00	X	1	V1, V5	0	1	1/0
2–mux	1/2	01	1	2	V1, V3, V5	0.354	0.791	2.236
2–mux	1/3	01	0	2	V1, V2, V4, V5	0.333	0.745	2.236
3–mux	1/2	10	1	3	V1, V3, V5	0.408	0.707	1.732
3–mux	1/3	10	0	3	V1, V2, V4, V5	0.333	0.638	1.915
4–mux	1/2	11	1	4	V1, V3, V5	0.433	0.661	1.528
4–mux	1/3	11	0	4	V1, V2, V4, V5	0.333	0.577	1.732

A typical approach to determine the required  $V_{LCD}$  is by equating  $V_{RMS,OFF}$  with a defined LCD threshold voltage, typically when the LCD exhibits approximately 10% contrast ( $V_{th,10\%}$ ):  $V_{RMS,OFF} = V_{th,10\%}$ . Using the values for  $V_{RMS,OFF}/V_{LCD}$  provided in the table results in  $V_{LCD} = V_{th,10\%}/(V_{RMS,OFF}/V_{LCD})$ . In the static mode a suitable choice is  $V_{LCD}$  greater or equal than 3 times  $V_{th,10\%}$ .

In 3-mux and 4-mux mode typically a 1/3 biasing is used but a 1/2 biasing scheme is also possible. The 1/2 bias reduces the contrast ratio but the advantage is a reduction of the required full-scale LCD voltage  $V_{LCD}$ .

### 19.2.4 LCD Timing Generation

The LCD\_A controller uses the  $f_{LCD}$  signal from the integrated ACLK prescaler to generate the timing for common and segment lines. ACLK is assumed to be 32768 Hz for generating  $f_{LCD}$ . The  $f_{LCD}$  frequency is selected with the LCDFREQx bits. The proper  $f_{LCD}$  frequency depends on the LCD's requirement for framing frequency and the LCD multiplex rate and is calculated by:

$$f_{LCD} = 2 \times \text{mux} \times f_{Frame}$$

For example, to calculate  $f_{LCD}$  for a 3-mux LCD, with a frame frequency of 30 - 100Hz:

$$f_{Frame} \text{ (from LCD datasheet)} = 30 - 100 \text{ Hz}$$

$$f_{LCD} = 2 \times 3 \times f_{Frame}$$

$$f_{LCD(\text{min})} = 180 \text{ Hz}$$

$$f_{LCD(\text{max})} = 600 \text{ Hz}$$

select  $f_{LCD} = 32768/128 = 256 \text{ Hz}$  or  $2768/96 = 341 \text{ Hz}$  or  $32768/64 = 512 \text{ Hz}$ .

The lowest frequency has the lowest current consumption. The highest frequency has the least flicker.

### 19.2.5 LCD Outputs

Some LCD segment, common, and Rxx functions are multiplexed with digital I/O functions. These pins can function either as digital I/O or as LCD functions. The pin functions for COMx and Rxx, when multiplexed with digital I/O, are selected using the applicable PxSELx bits as described in the *Digital I/O* chapter. The LCD segment functions, when multiplexed with digital I/O, are selected using the LCDSx bits in the LCDAPCTLx registers.

The LCDSx bits selects the LCD function in groups of four pins. When LCDSx = 0, no multiplexed pin is set to LCD function. When LCDSx = 1, the complete group of four is selected as LCD function.

**Note: LCDSx Bits Do Not Affect Dedicated LCD Segment Pins**

The LCDSx bits only affect pins with multiplexed LCD segment functions and digital I/O functions. Dedicated LCD segment pins are not affected by the LCDSx bits.

### 19.2.6 Static Mode

In static mode, each MSP430 segment pin drives one LCD segment and one common line, COM0, is used. Figure 19–4 shows some example static waveforms.

Figure 19–4. Example Static Waveforms

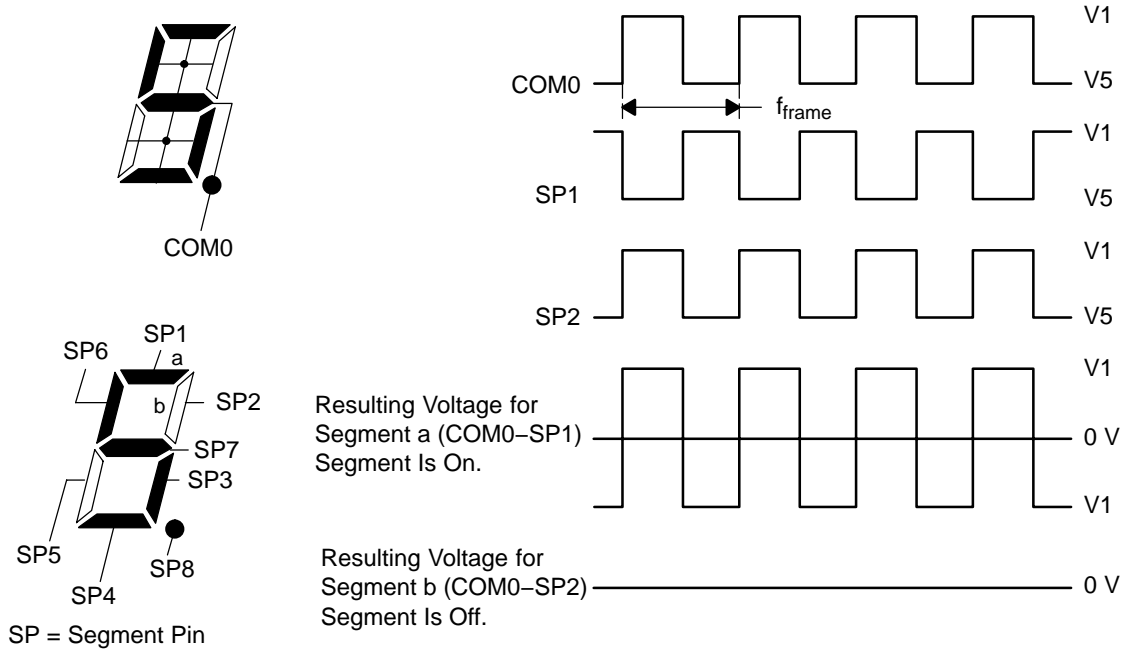
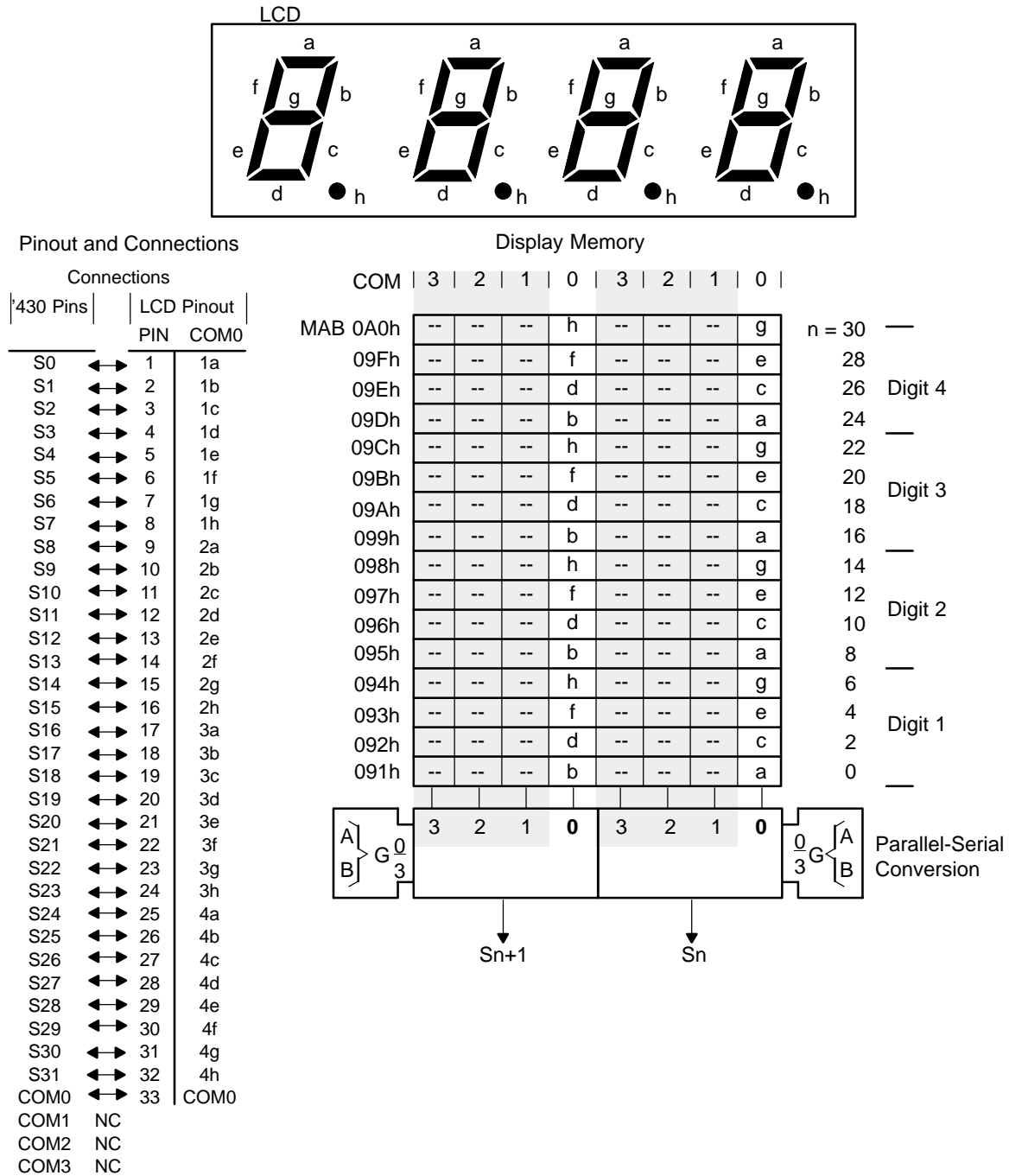


Figure 19–5 shows an example static LCD, pin-out, LCD-to-MSP430 connections, and the resulting segment mapping. This is only an example. Segment mapping in a user’s application depends on the LCD pin-out and on the MSP430-to-LCD connections.

Figure 19–5. Static LCD Example





## Static Mode Software Example

```

; All eight segments of a digit are often located in four
; display memory bytes with the static display method.
;
a    EQU    001h
b    EQU    010h
c    EQU    002h
d    EQU    020h
e    EQU    004h
f    EQU    040h
g    EQU    008h
h    EQU    080h
; The register content of Rx should be displayed.
; The Table represents the 'on'-segments according to the
; content of Rx.
    MOV.B Table (Rx),RY    ; Load segment information
                           ; into temporary memory.
                           ; (Ry) = 0000 0000 hfdb geca
    MOV.B Ry,&LCDn         ; Note:
                           ; All bits of an LCD memory
                           ; byte are written
    RRA Ry                ; (Ry) = 0000 0000 0hfd bgec
    MOV.B Ry,&LCDn+1       ; Note:
                           ; All bits of an LCD memory
                           ; byte are written
    RRA Ry                ; (Ry) = 0000 0000 00hf dbge
    MOV.B Ry,&LCDn+2       ; Note:
                           ; All bits of an LCD memory
                           ; byte are written
    RRA Ry                ; (Ry) = 0000 0000 000h fdbg
    MOV.B Ry,&LCDn+3       ; Note:
                           ; All bits of an LCD memory
                           ; byte are written

    .....
    .....
;
Table DB    a+b+c+d+e+f    ; displays "0"
      DB    b+c;          ; displays "1"
      .....
      .....
      DB
      .....

```

### 19.2.7 2-Mux Mode

In 2-mux mode, each MSP430 segment pin drives two LCD segments and two common lines, COM0 and COM1, are used. Figure 19–6 shows some example 2-mux, 1/2 bias waveforms.

Figure 19–6. Example 2-Mux Waveforms

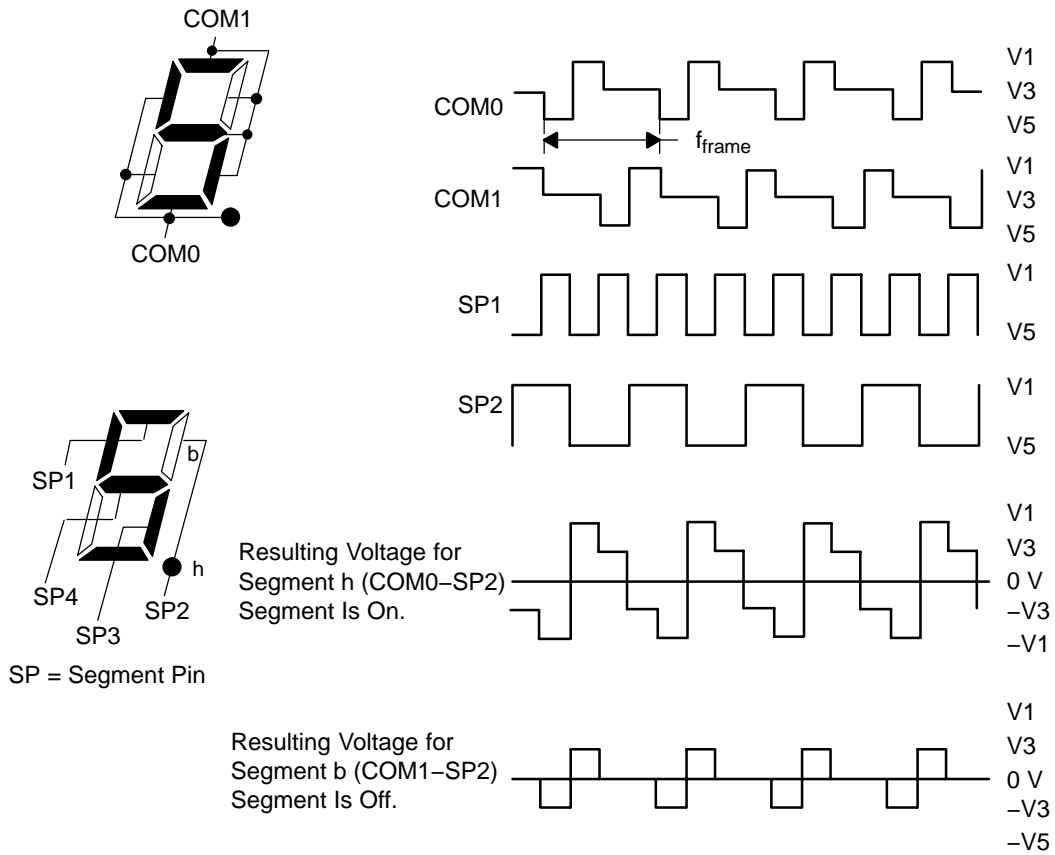
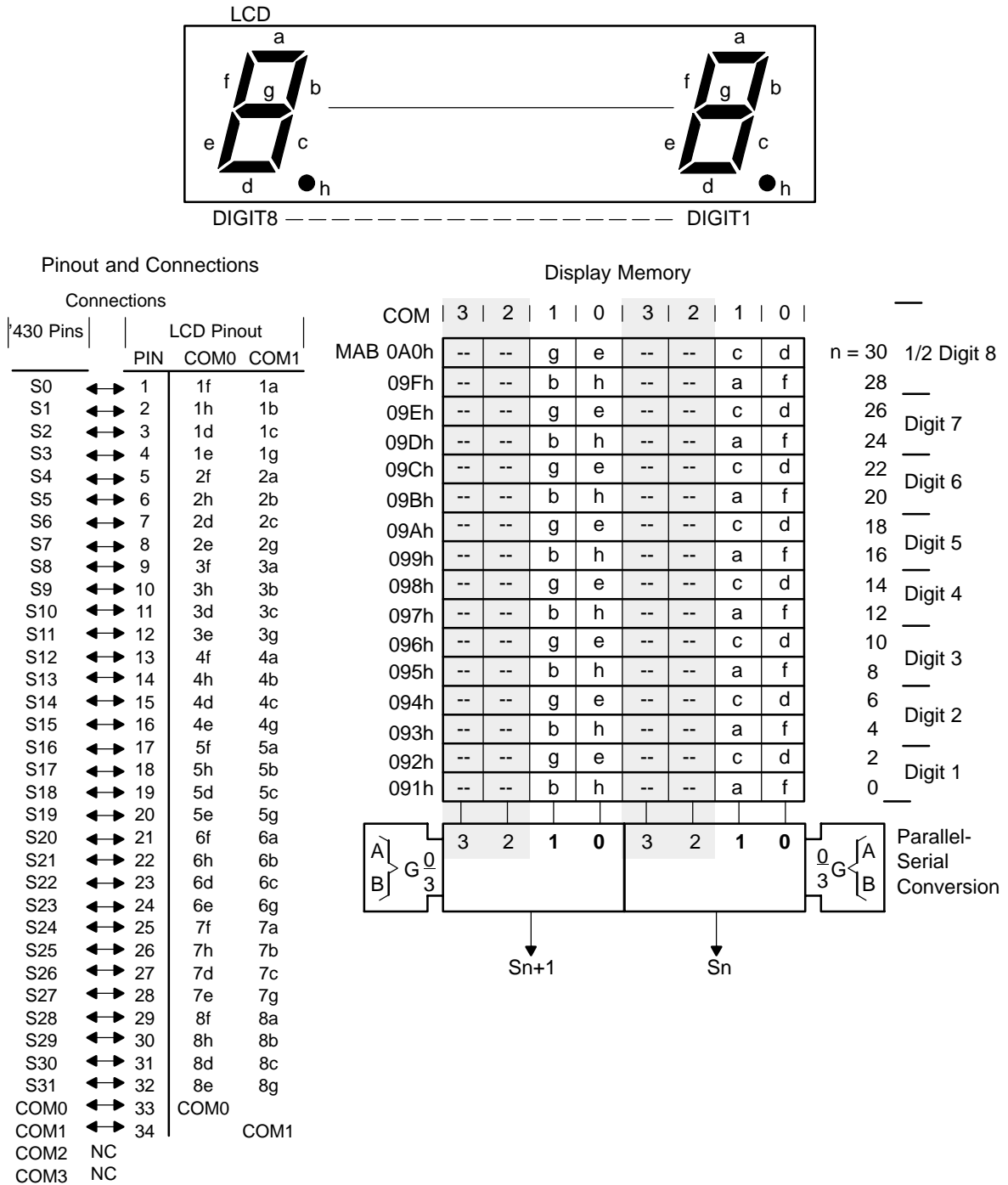


Figure 19–7 shows an example 2-mux LCD, pin-out, LCD-to-MSP430 connections, and the resulting segment mapping. This is only an example. Segment mapping in a user’s application completely depends on the LCD pin-out and on the MSP430-to-LCD connections.

Figure 19–7. 2-Mux LCD Example



## 2-Mux Mode Software Example

```

; All eight segments of a digit are often located in two
; display memory bytes with the 2mux display rate
;
a    EQU    002h
b    EQU    020h
c    EQU    008h
d    EQU    004h
e    EQU    040h
f    EQU    001h
g    EQU    080h
h    EQU    010h
; The register content of Rx should be displayed.
; The Table represents the 'on'-segments according to the
; content of Rx.
;
.....
.....
MOV.B Table(Rx),Ry ; Load segment information into
                   ; temporary memory.
MOV.B Ry,&LCDn      ; (Ry) = 0000 0000 gebh cdaf
                   ; Note:
                   ; All bits of an LCD memory byte
                   ; are written
RRA   Ry           ; (Ry) = 0000 0000 0geb hcda
RRA   Ry           ; (Ry) = 0000 0000 00ge bhcd
MOV.B Ry,&LCDn+1   ; Note:
                   ; All bits of an LCD memory byte
                   ; are written
.....
.....
.....
Table DB    a+b+c+d+e+f      ; displays "0"
.....
DB    a+b+c+d+e+f+g+       ; displays "8"
.....
.....
DB
.....
;

```

### 19.2.8 3-Mux Mode

In 3-mux mode, each MSP430 segment pin drives three LCD segments and three common lines, COM0, COM1 and COM2 are used. Figure 19–8 shows some example 3-mux, 1/3 bias waveforms.

Figure 19–8. Example 3-Mux Waveforms

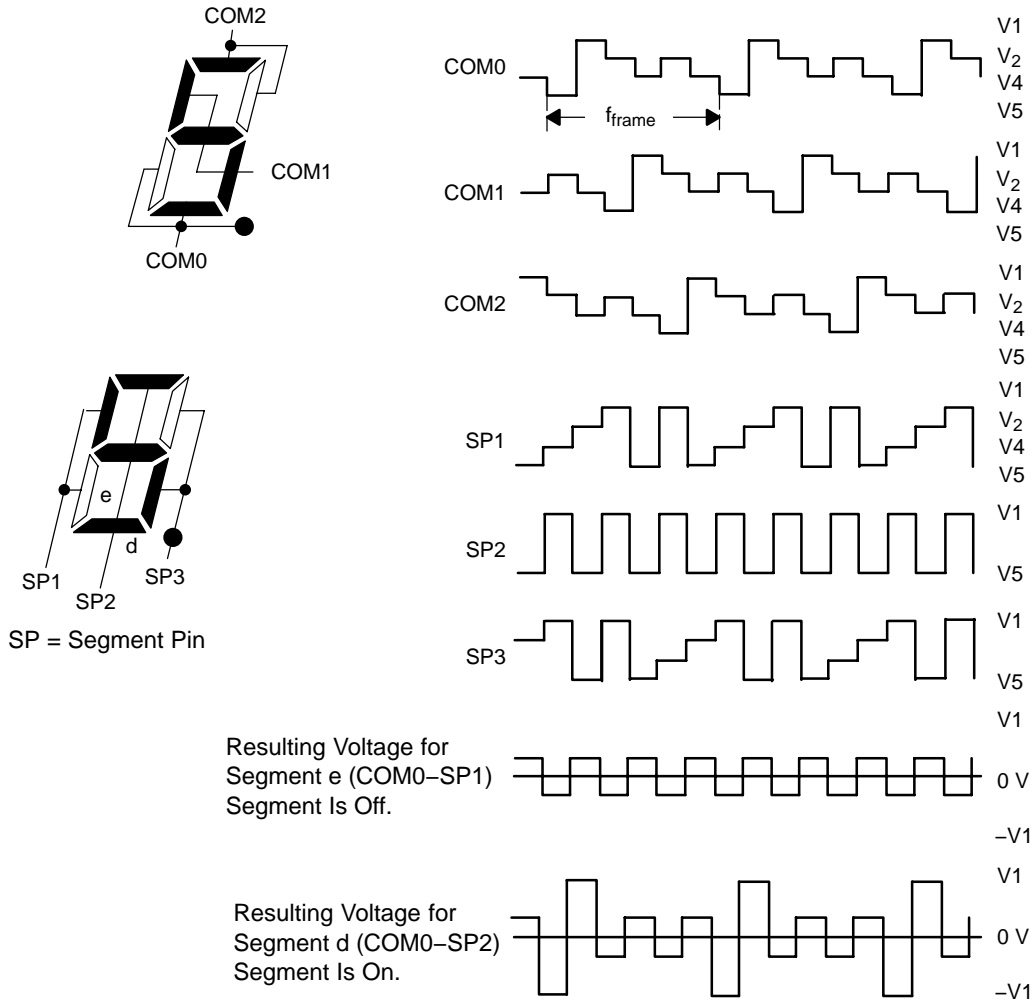
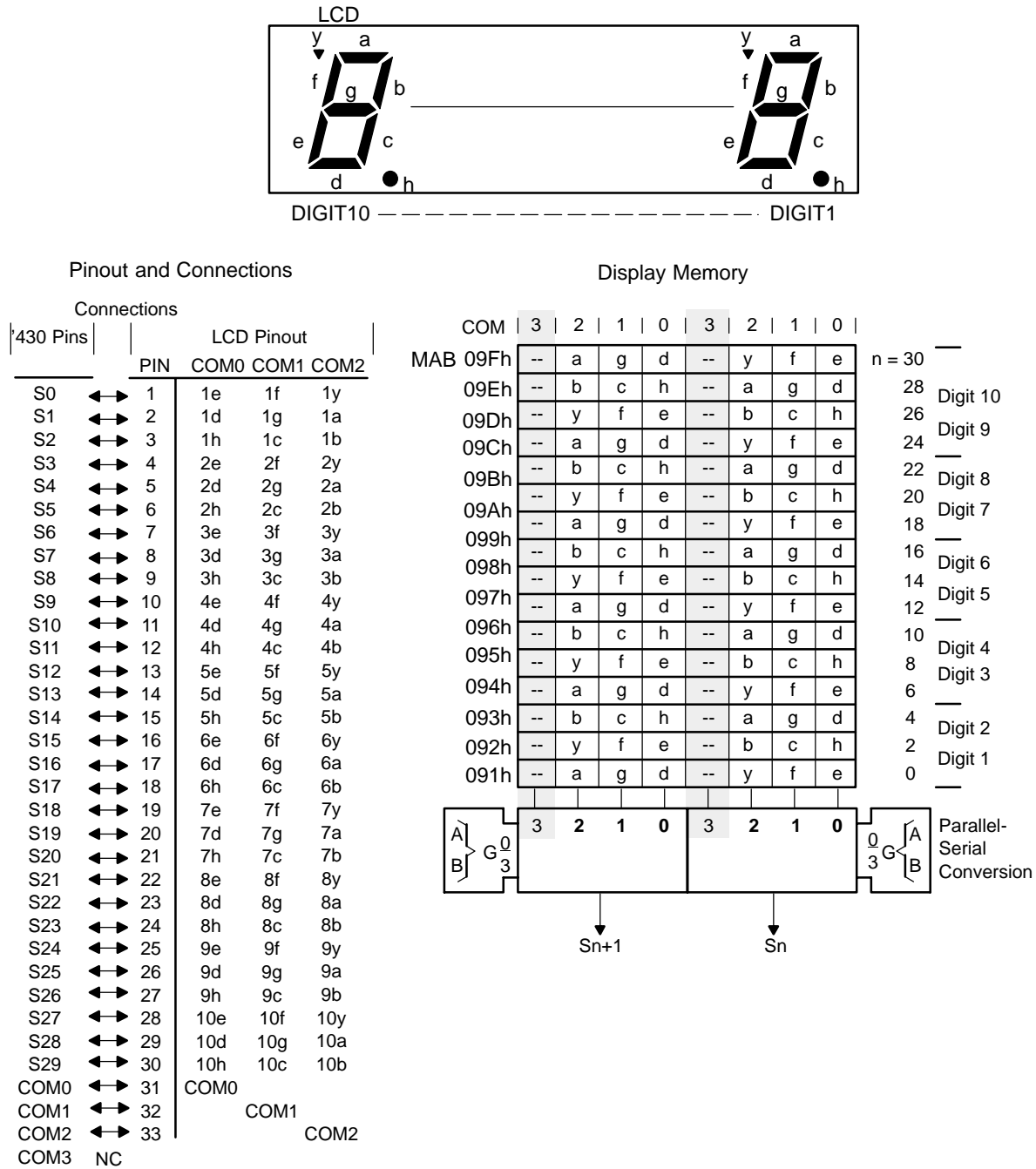


Figure 19–9 shows an example 3-mux LCD, pin-out, LCD-to-MSP430 connections, and the resulting segment mapping. This is only an example. Segment mapping in a user’s application depends on the LCD pin-out and on the MSP430-to-LCD connections.

Figure 19–9. 3-Mux LCD Example



### 3-Mux Mode Software Example

```

; The 3mux rate can support nine segments for each
; digit. The nine segments of a digit are located in
; 1 1/2 display memory bytes.
;
a    EQU    0040h
b    EQU    0400h
c    EQU    0200h
d    EQU    0010h
e    EQU    0001h
f    EQU    0002h
g    EQU    0020h
h    EQU    0100h
Y    EQU    0004h
; The LSDigit of register Rx should be displayed.
; The Table represents the 'on'-segments according to the
; LSDigit of register of Rx.
; The register Ry is used for temporary memory
;
ODDDIGRLA    Rx                ; LCD in 3mux has 9 segments per
; digit; word table required for
; displayed characters.
        MOV    Table(Rx),Ry    ; Load segment information to
; temporary mem.
; (Ry) = 0000 0bch 0agd 0yfe
        MOV.B  Ry,&LCDn        ; write 'a, g, d, y, f, e' of
; Digit n (LowByte)
        SWPB  Ry                ; (Ry) = 0agd 0yfe 0000 0bch
        BIC.B #07h,&LCDn+1    ; write 'b, c, h' of Digit n
; (HighByte)
        BIS.B Ry,&LCDn+1
        .....
EVNDIGRLA    Rx                ; LCD in 3mux has 9 segments per
; digit; word table required for
; displayed characters.
        MOV    Table(Rx),Ry    ; Load segment information to
; temporary mem.
; (Ry) = 0000 0bch 0agd 0yfe
        RLA   Ry                ; (Ry) = 0000 bch0 agd0 yfe0
        RLA   Ry                ; (Ry) = 000b ch0a gd0y fe00
        RLA   Ry                ; (Ry) = 00bc h0ag d0yf e000
        RLA   Ry                ; (Ry) = 0bch 0agd 0yfe 0000
        BIC.B #070h,&LCDn+1
        BIS.B Ry,&LCDn+1        ; write 'y, f, e' of Digit n+1
; (LowByte)
        SWPB  Ry                ; (Ry) = 0yfe 0000 0bch 0agd
        MOV.B Ry,&LCDn+2        ; write 'b, c, h, a, g, d' of
; Digit n+1 (HighByte)
        .....
Table    DW    a+b+c+d+e+f    ; displays "0"
        DW    b+c                ; displays "1"
        .....
        DW    a+e+f+g        ; displays "F"

```

### 19.2.9 4-Mux Mode

In 3-mux mode, each MSP430 segment pin drives four LCD segments and all four common lines, COM0, COM1, COM2, and COM3 are used. Figure 19–10 shows some example 4-mux, 1/3 bias waveforms.

Figure 19–10. Example 4-Mux Waveforms

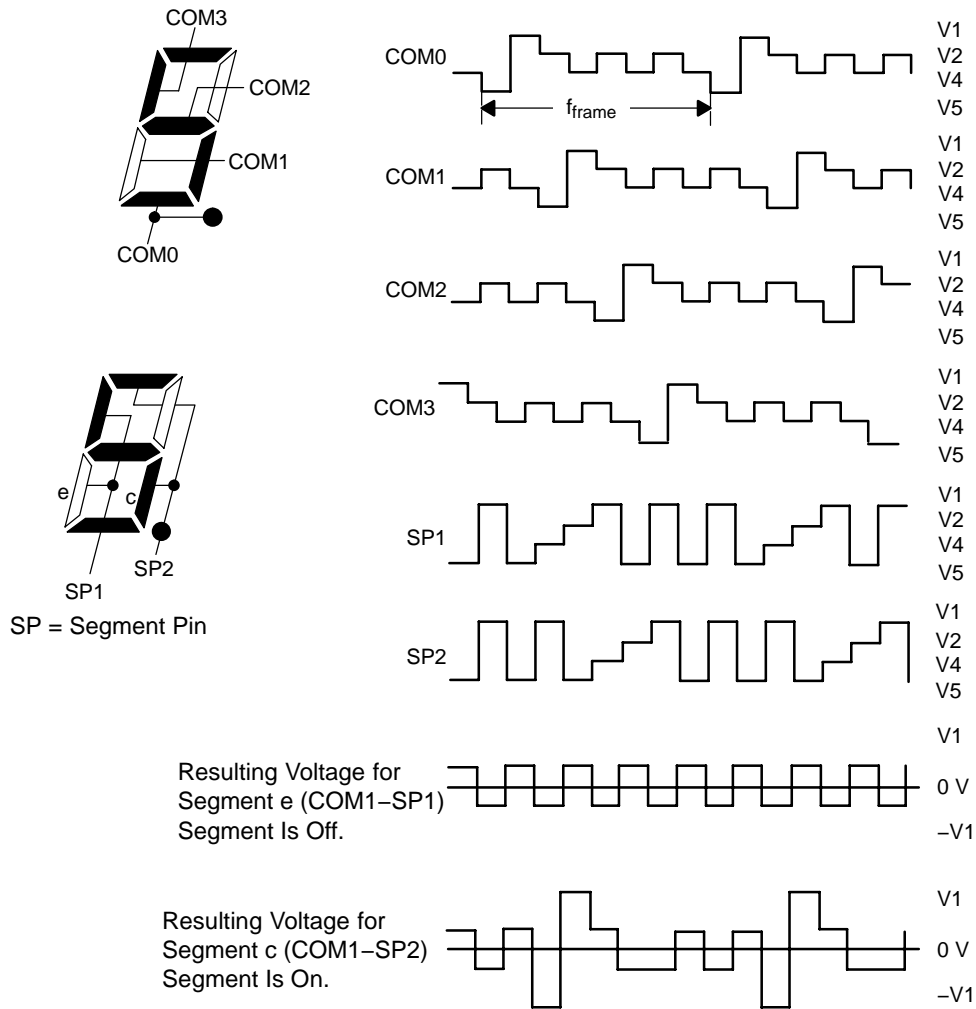
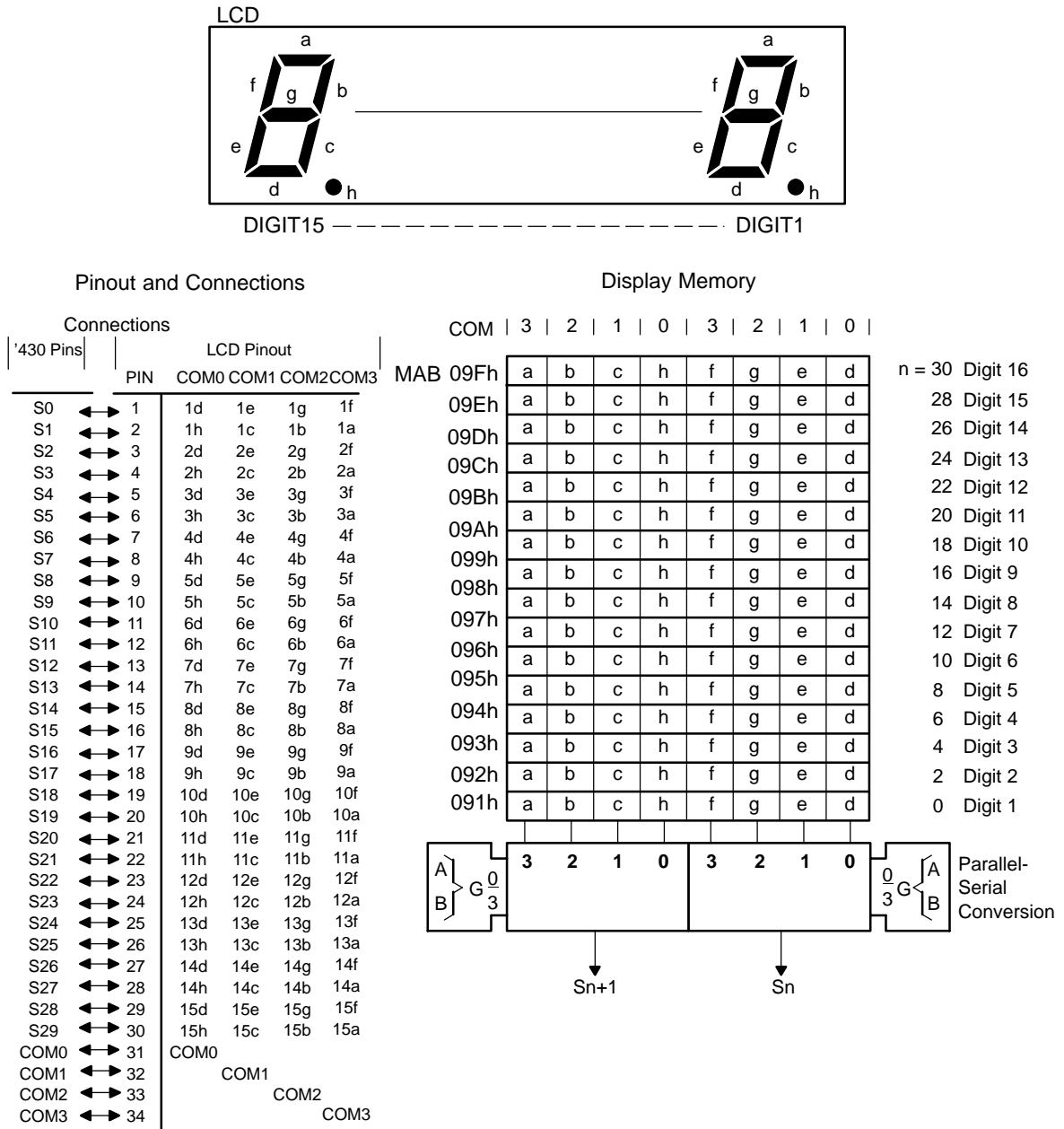




Figure 19–11 shows an example 4-mux LCD, pin-out, LCD-to-MSP430 connections, and the resulting segment mapping. This is only an example. Segment mapping in a user’s application depends on the LCD pin-out and on the MSP430-to-LCD connections.

Figure 19–11. 4-Mux LCD Example



## 4-Mux Mode Software Example

```

; The 4mux rate supports eight segments for each digit.
; All eight segments of a digit can often be located in
; one display memory byte
a EQU 080h
b EQU 040h
c EQU 020h
d EQU 001h
e EQU 002h
f EQU 008h
g EQU 004h
h EQU 010h
;
; The LSDigit of register Rx should be displayed.
; The Table represents the 'on'-segments according to the
; content of Rx.
;
MOV.B Table(Rx),&LCDn ; n = 1 ..... 15
                        ; all eight segments are
                        ; written to the display
                        ; memory
.....
.....
Table DB a+b+c+d+e+f ; displays "0"
      DB b+c         ; displays "1"
      .....
      .....
      DB b+c+d+e+g   ; displays "d"
      DB a+d+e+f+g   ; displays "E"
      DB a+e+f+g     ; displays "F"

```

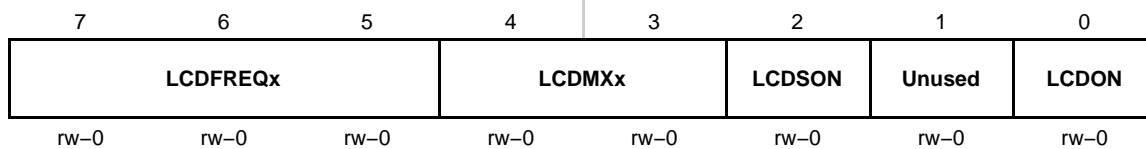
### 19.3 LCD Controller Registers

The LCD Controller registers are listed in Table 19–2.

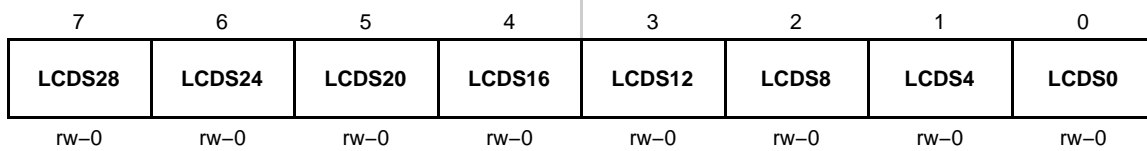
Table 19–2. LCD Controller Registers

Register	Short Form	Register Type	Address	Initial State
LCD_A control register	LCDACTL	Read/write	090h	Reset with PUC
LCD memory 1	LCDM1	Read/write	091h	Unchanged
LCD memory 2	LCDM2	Read/write	092h	Unchanged
LCD memory 3	LCDM3	Read/write	093h	Unchanged
LCD memory 4	LCDM4	Read/write	094h	Unchanged
LCD memory 5	LCDM5	Read/write	095h	Unchanged
LCD memory 6	LCDM6	Read/write	096h	Unchanged
LCD memory 7	LCDM7	Read/write	097h	Unchanged
LCD memory 8	LCDM8	Read/write	098h	Unchanged
LCD memory 9	LCDM9	Read/write	099h	Unchanged
LCD memory 10	LCDM10	Read/write	09Ah	Unchanged
LCD memory 11	LCDM11	Read/write	09Bh	Unchanged
LCD memory 12	LCDM12	Read/write	09Ch	Unchanged
LCD memory 13	LCDM13	Read/write	09Dh	Unchanged
LCD memory 14	LCDM14	Read/write	09Eh	Unchanged
LCD memory 15	LCDM15	Read/write	09Fh	Unchanged
LCD memory 16	LCDM16	Read/write	0A0h	Unchanged
LCD memory 17	LCDM17	Read/write	0A1h	Unchanged
LCD memory 18	LCDM18	Read/write	0A2h	Unchanged
LCD memory 19	LCDM19	Read/write	0A3h	Unchanged
LCD memory 20	LCDM20	Read/write	0A4h	Unchanged
LCD_A port control 0	LCDA PCTL0	Read/write	0AC h	Reset with PUC
LCD_A port control 1	LCDA PCTL1	Read/write	0AD h	Reset with PUC
LCD_A voltage control 0	LCDA VCTL0	Read/write	0AE h	Reset with PUC
LCD_A voltage control 1	LCDA VCTL1	Read/write	0AF h	Reset with PUC

## LCDACTL, LCD\_A Control Register

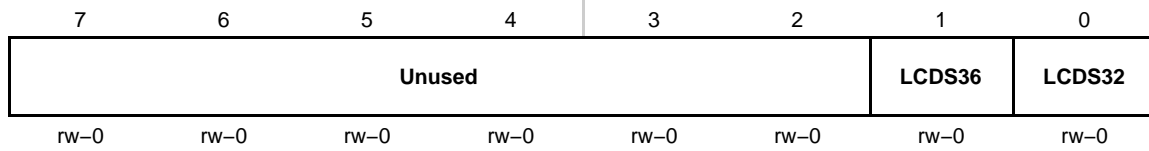


<b>LCDFREQx</b>	Bits 7-5	LCD Frequency Select. These bits select the ACLK divider for the LCD frequency. 000 Divide by 32 001 Divide by 64 010 Divide by 96 011 Divide by 128 100 Divide by 192 101 Divide by 256 110 Divide by 384 111 Divide by 512
<b>LCDMXx</b>	Bits 4-3	LCD mux rate. These bits select the LCD mode. 00 Static 01 2-mux 10 3-mux 11 4-mux
<b>LCDSON</b>	Bit 2	LCD segments on. This bit supports flashing LCD applications by turning off all segment lines, while leaving the LCD timing generator and R33 enabled. 0 All LCD segments are off 1 All LCD segments are enabled and on or off according to their corresponding memory location.
<b>Unused</b>	Bit 1	Unused
<b>LCDON</b>	Bit 0	LCD On. This bit turns on the LCD_A module. 0 LCD_A module off. 1 LCD_A module on.

**LCDAPCTL0, LCD\_A Port Control Register 0**

<b>LCDS28</b>	Bit 7	<p>LCD Segment 28 to 31 Enable. This bit only affects pins with multiplexed functions. Dedicated LCD pins are always LCD function.</p> <p>0 Multiplexed pins are port functions. 1 Pins are LCD functions</p>
<b>LCDS24</b>	Bit 6	<p>LCD Segment 24 to 27 Enable. This bit only affects pins with multiplexed functions. Dedicated LCD pins are always LCD function.</p> <p>0 Multiplexed pins are port functions. 1 Pins are LCD functions</p>
<b>LCDS20</b>	Bit 5	<p>LCD Segment 20 to 23 Enable. This bit only affects pins with multiplexed functions. Dedicated LCD pins are always LCD function.</p> <p>0 Multiplexed pins are port functions. 1 Pins are LCD functions</p>
<b>LCDS16</b>	Bit 4	<p>LCD Segment 16 to 19 Enable. This bit only affects pins with multiplexed functions. Dedicated LCD pins are always LCD function.</p> <p>0 Multiplexed pins are port functions. 1 Pins are LCD functions</p>
<b>LCDS12</b>	Bit 3	<p>LCD Segment 12 to 13 Enable. This bit only affects pins with multiplexed functions. Dedicated LCD pins are always LCD function.</p> <p>0 Multiplexed pins are port functions. 1 Pins are LCD functions</p>
<b>LCDS8</b>	Bit 2	<p>LCD Segment 8 to 11 Enable. This bit only affects pins with multiplexed functions. Dedicated LCD pins are always LCD function.</p> <p>0 Multiplexed pins are port functions. 1 Pins are LCD functions</p>
<b>LCDS4</b>	Bit 1	<p>LCD Segment 4 to 7 Enable. This bit only affects pins with multiplexed functions. Dedicated LCD pins are always LCD function.</p> <p>0 Multiplexed pins are port functions. 1 Pins are LCD functions</p>
<b>LCDS0</b>	Bit 0	<p>LCD Segment 0 to 3 Enable. This bit only affects pins with multiplexed functions. Dedicated LCD pins are always LCD function.</p> <p>0 Multiplexed pins are port functions. 1 Pins are LCD functions</p>

### LCDACTL1, LCD\_A Port Control Register 1



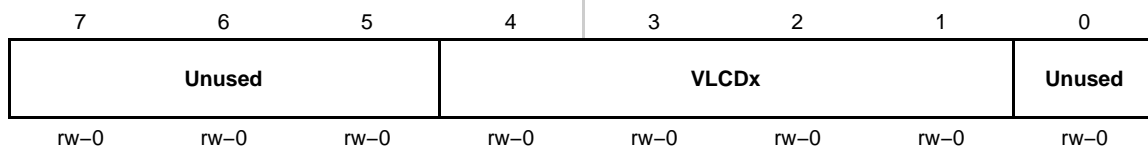
<b>Unused</b>	Bits 7-2	Unused
<b>LCDS36</b>	Bit 1	LCD Segment 36 to 39 Enable. This bit only affects pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0 Multiplexed pins are port functions. 1 Pins are LCD functions
<b>LCDS32</b>	Bit 0	LCD Segment 32 to 35 Enable. This bit only affects pins with multiplexed functions. Dedicated LCD pins are always LCD function. 0 Multiplexed pins are port functions. 1 Pins are LCD functions

## LCDAVCTL0, LCD\_A Voltage Control Register 0

7	6	5	4	3	2	1	0
Unused	R03EXT	REXT	VLCDEXT	LCDCPEN	VLCDFREFx		LCD2B
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

<b>Unused</b>	Bit 7	Unused
<b>R03EXT</b>	Bit 6	V5 voltage select. This bit selects the external connection for the lowest LCD voltage. R03EXT is ignored if there is no R03 pin available. 0 V5 is AV <sub>CC</sub> 1 V5 is sourced from the R03 pin
<b>REXT</b>	Bit 5	V2 – V4 voltage select. This bit selects the external connections for voltages V2 – V4. 0 V2 – V4 are generated internally 1 V2 – V4 are sourced externally and the internal bias generator is switched off
<b>VLCDEXT</b>	Bit 4	V <sub>LCD</sub> source select 0 V <sub>LCD</sub> is generated internally 1 V <sub>LCD</sub> is sourced externally
<b>LCDCPEN</b>	Bit 3	Charge pump enable. 0 Charge pump disabled. 1 Charge pump enabled when VLCD <sub>x</sub> > 0 or VLCDFREF <sub>x</sub> > 0 and VLCDEXT = 0
<b>VLCDFREFx</b>	Bits 2–1	Charge pump reference select 00 Internal 01 External 10 Reserved 11 Reserved
<b>LCD2B</b>	Bit 0	Bias select. LCD2B is ignored when LCDM <sub>x</sub> = 00. 0 1/3 bias 1 1/2 bias

**LCDAVCTL1, LCD\_A Voltage Control Register 1**



<b>Unused</b>	Bits 7-5	Unused
<b>VLCDx</b>	Bits 4-1	<p>Charge pump voltage select. LCDPEN must be 1 for the charge pump to be enabled. AV<sub>CC</sub> is used for V<sub>LCD</sub> when VLCDx = 0000 and VREFx = 00 and VLCDEXT = 0.</p> <p>0000 Charge pump disabled.</p> <p>0001 V<sub>LCD</sub> = 2.60V</p> <p>0010 V<sub>LCD</sub> = 2.66V</p> <p>0011 V<sub>LCD</sub> = 2.72V</p> <p>0100 V<sub>LCD</sub> = 2.78V</p> <p>0101 V<sub>LCD</sub> = 2.84V</p> <p>0110 V<sub>LCD</sub> = 2.90V</p> <p>0111 V<sub>LCD</sub> = 2.96V</p> <p>1000 V<sub>LCD</sub> = 3.02V</p> <p>1001 V<sub>LCD</sub> = 3.08V</p> <p>1010 V<sub>LCD</sub> = 3.14V</p> <p>1011 V<sub>LCD</sub> = 3.20V</p> <p>1100 V<sub>LCD</sub> = 3.26V</p> <p>1101 V<sub>LCD</sub> = 3.32V</p> <p>1110 V<sub>LCD</sub> = 3.38V</p> <p>1111 V<sub>LCD</sub> = 3.44V</p>
<b>Unused</b>	Bit 0	Unused



**ADC12**

---

---

---

---

The ADC12 module is a high-performance 12-bit analog-to-digital converter. This chapter describes the ADC12. The ADC12 is implemented in the MSP430x43x and MSP430x44x devices.

<b>Topic</b>	<b>Page</b>
<b>20.1 ADC12 Introduction</b> .....	<b>20-2</b>
<b>20.2 ADC12 Operation</b> .....	<b>20-4</b>
<b>20.3 ADC12 Registers</b> .....	<b>20-20</b>

## 20.1 ADC12 Introduction

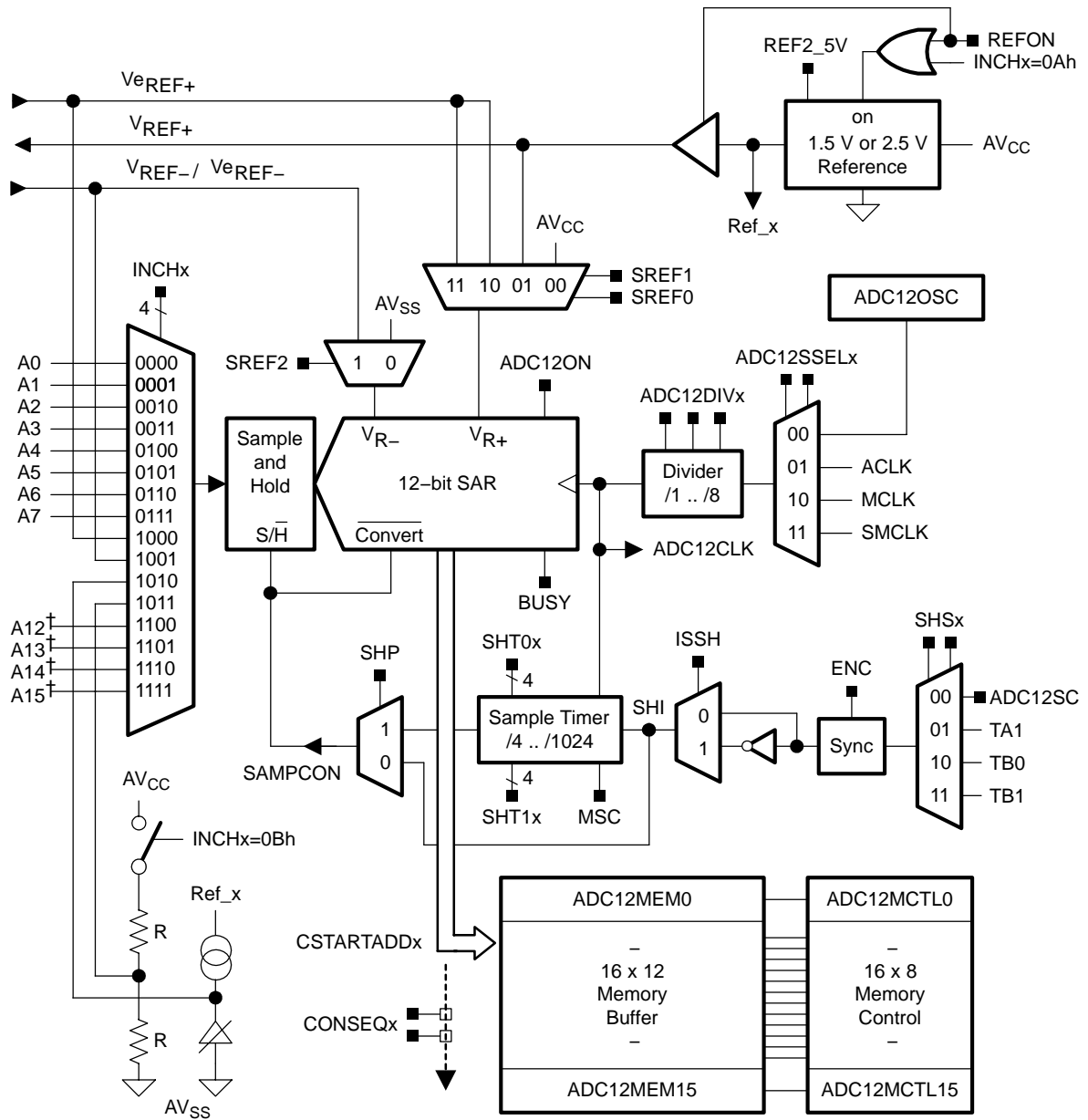
The ADC12 module supports fast, 12-bit analog-to-digital conversions. The module implements a 12-bit SAR core, sample select control, reference generator and a 16 word conversion-and-control buffer. The conversion-and-control buffer allows up to 16 independent ADC samples to be converted and stored without any CPU intervention.

ADC12 features include:

- Greater than 200 ksps maximum conversion rate
- Monotonic 12-bit converter with no missing codes
- Sample-and-hold with programmable sampling periods controlled by software or timers.
- Conversion initiation by software, Timer\_A, or Timer\_B
- Software selectable on-chip reference voltage generation (1.5 V or 2.5 V)
- Software selectable internal or external reference
- Eight individually configurable external input channels (twelve on MSP430FG43x devices)
- Conversion channels for internal temperature sensor,  $AV_{CC}$ , and external references
- Independent channel-selectable reference sources for both positive and negative references
- Selectable conversion clock source
- Single-channel, repeat-single-channel, sequence, and repeat-sequence conversion modes
- ADC core and reference voltage can be powered down separately
- Interrupt vector register for fast decoding of 18 ADC interrupts
- 16 conversion-result storage registers

The block diagram of ADC12 is shown in Figure 20–1.

Figure 20-1. ADC12 Block Diagram



†MSP430FG43x devices only

## 20.2 ADC12 Operation

The ADC12 module is configured with user software. The setup and operation of the ADC12 is discussed in the following sections.

### 20.2.1 12-Bit ADC Core

The ADC core converts an analog input to its 12-bit digital representation and stores the result in conversion memory. The core uses two programmable/selectable voltage levels ( $V_{R+}$  and  $V_{R-}$ ) to define the upper and lower limits of the conversion. The digital output ( $N_{ADC}$ ) is full scale (0FFFh) when the input signal is equal to or higher than  $V_{R+}$ , and zero when the input signal is equal to or lower than  $V_{R-}$ . The input channel and the reference voltage levels ( $V_{R+}$  and  $V_{R-}$ ) are defined in the conversion-control memory. The conversion formula for the ADC result  $N_{ADC}$  is:

$$N_{ADC} = 4095 \times \frac{V_{in} - V_{R-}}{V_{R+} - V_{R-}}$$

The ADC12 core is configured by two control registers, ADC12CTL0 and ADC12CTL1. The core is enabled with the ADC12ON bit. The ADC12 can be turned off when not in use to save power. With few exceptions the ADC12 control bits can only be modified when ENC = 0. ENC must be set to 1 before any conversion can take place.

### Conversion Clock Selection

The ADC12CLK is used both as the conversion clock and to generate the sampling period when the pulse sampling mode is selected. The ADC12 source clock is selected using the ADC12SELx bits and can be divided from 1-8 using the ADC12DIVx bits. Possible ADC12CLK sources are SMCLK, MCLK, ACLK, and an internal oscillator ADC12OSC.

The ADC12OSC, generated internally, is in the 5-MHz range, but varies with individual devices, supply voltage, and temperature. See the device-specific datasheet for the ADC12OSC specification.

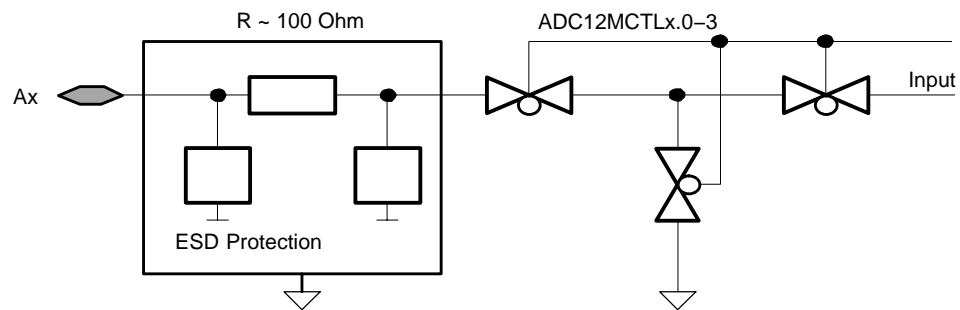
The user must ensure that the clock chosen for ADC12CLK remains active until the end of a conversion. If the clock is removed during a conversion, the operation will not complete and any result will be invalid.

## 20.2.2 ADC12 Inputs and Multiplexer

The eight external and four internal analog signals are selected as the channel for conversion by the analog input multiplexer. The input multiplexer is a break-before-make type to reduce input-to-input noise injection resulting from channel switching as shown in Figure 20–2. The input multiplexer is also a T-switch to minimize the coupling between channels. Channels that are not selected are isolated from the A/D and the intermediate node is connected to analog ground ( $AV_{SS}$ ) so that the stray capacitance is grounded to help eliminate crosstalk.

The ADC12 uses the charge redistribution method. When the inputs are internally switched, the switching action may cause transients on the input signal. These transients decay and settle before causing errant conversion.

Figure 20–2. Analog Multiplexer



### Analog Port Selection

The ADC12 inputs are multiplexed with the port P6 pins, which are digital CMOS gates. When analog signals are applied to digital CMOS gates, parasitic current can flow from  $V_{CC}$  to GND. This parasitic current occurs if the input voltage is near the transition level of the gate. Disabling the port pin buffer eliminates the parasitic current flow and therefore reduces overall current consumption. The P6SELx bits provide the ability to disable the port pin input and output buffers.

```
; P6.0 and P6.1 configured for analog input
    BIS.B #3h,&P6SEL ; P6.1 and P6.0 ADC12 function
```

### 20.2.3 Voltage Reference Generator

The ADC12 module contains a built-in voltage reference with two selectable voltage levels, 1.5 V and 2.5 V. Either of these reference voltages may be used internally and externally on pin  $V_{REF+}$ .

Setting  $REFON=1$  enables the internal reference. When  $REF2\_5V = 1$ , the internal reference is 2.5 V, the reference is 1.5 V when  $REF2\_5V = 0$ . The reference can be turned off to save power when not in use.

For proper operation the internal voltage reference generator must be supplied with storage capacitance across  $V_{REF+}$  and  $A_{VSS}$ . The recommended storage capacitance is a parallel combination of 10- $\mu$ F and 0.1- $\mu$ F capacitors. From turn-on, a maximum of 17 ms must be allowed for the voltage reference generator to bias the recommended storage capacitors. If the internal reference generator is not used for the conversion, the storage capacitors are not required.

**Note: Reference Decoupling**

Approximately 200  $\mu$ A is required from *any* reference used by the ADC12 while the two LSBs are being resolved during a conversion. A parallel combination of 10- $\mu$ F and 0.1- $\mu$ F capacitors is recommended for *any* reference used as shown in Figure 20–11.

External references may be supplied for  $V_{R+}$  and  $V_{R-}$  through pins  $Ve_{REF+}$  and  $V_{REF-}/Ve_{REF-}$  respectively.

### 20.2.4 Auto Power-Down

The ADC12 is designed for low power applications. When the ADC12 is not actively converting, the core is automatically disabled and automatically re-enabled when needed. The ADC12OSC is also automatically enabled when needed and disabled when not needed. The reference is not automatically disabled, but can be disabled by setting  $REFON = 0$ . When the core, oscillator, or reference are disabled, they consume no current.

## 20.2.5 Sample and Conversion Timing

An analog-to-digital conversion is initiated with a rising edge of the sample input signal SHI. The source for SHI is selected with the SHSx bits and includes the following:

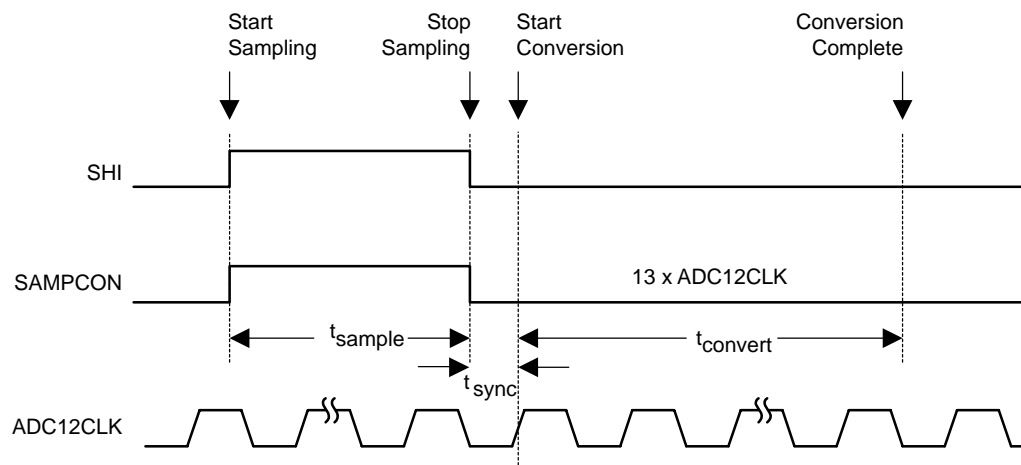
- The ADC12SC bit
- The Timer\_A Output Unit 1
- The Timer\_B Output Unit 0
- The Timer\_B Output Unit 1

The polarity of the SHI signal source can be inverted with the ISSH bit. The SAMPCON signal controls the sample period and start of conversion. When SAMPCON is high, sampling is active. The high-to-low SAMPCON transition starts the analog-to-digital conversion, which requires 13 ADC12CLK cycles. Two different sample-timing methods are defined by control bit SHP, extended sample mode and pulse mode.

### Extended Sample Mode

The extended sample mode is selected when SHP = 0. The SHI signal directly controls SAMPCON and defines the length of the sample period  $t_{\text{sample}}$ . When SAMPCON is high, sampling is active. The high-to-low SAMPCON transition starts the conversion after synchronization with ADC12CLK. See Figure 20–3.

Figure 20–3. Extended Sample Mode

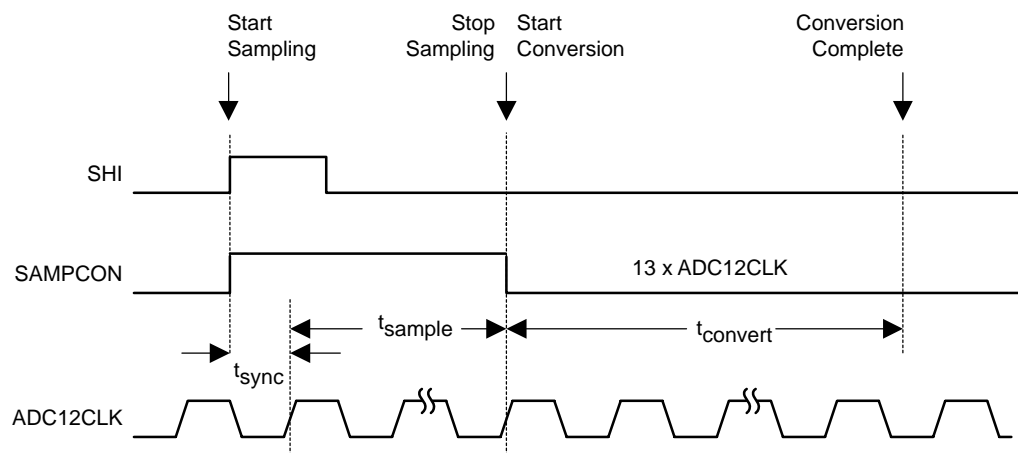


## Pulse Sample Mode

The pulse sample mode is selected when  $SHP = 1$ . The SHI signal is used to trigger the sampling timer. The SHT0x and SHT1x bits in ADC12CTL0 control the interval of the sampling timer that defines the SAMPCON sample period  $t_{\text{sample}}$ . The sampling timer keeps SAMPCON high after synchronization with AD12CLK for a programmed interval  $t_{\text{sample}}$ . The total sampling time is  $t_{\text{sample}}$  plus  $t_{\text{sync}}$ . See Figure 20–4.

The SHTx bits select the sampling time in 4x multiples of ADC12CLK. SHT0x selects the sampling time for ADC12MCTL0 to 7 and SHT1x selects the sampling time for ADC12MCTL8 to 15.

Figure 20–4. Pulse Sample Mode

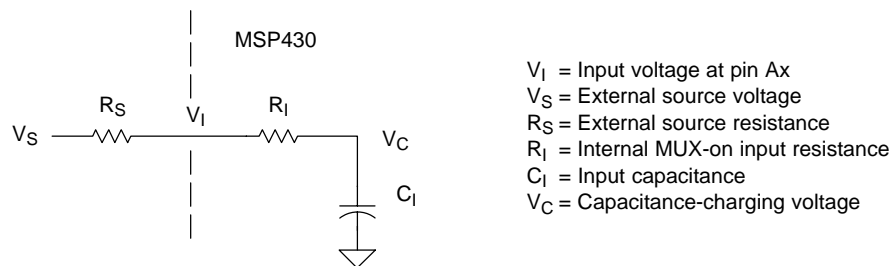




## Sample Timing Considerations

When  $\text{SAMPCON} = 0$  all  $A_x$  inputs are high impedance. When  $\text{SAMPCON} = 1$ , the selected  $A_x$  input can be modeled as an RC low-pass filter during the sampling time  $t_{\text{sample}}$ , as shown below in Figure 20–5. An internal MUX-on input resistance  $R_I$  (max.  $2\text{ k}\Omega$ ) in series with capacitor  $C_I$  (max.  $40\text{ pF}$ ) is seen by the source. The capacitor  $C_I$  voltage  $V_C$  must be charged to within  $1/2$  LSB of the source voltage  $V_S$  for an accurate 12-bit conversion.

Figure 20–5. Analog Input Equivalent Circuit



The resistance of the source  $R_S$  and  $R_I$  affect  $t_{\text{sample}}$ . The following equation can be used to calculate the minimum sampling time  $t_{\text{sample}}$  for a 12-bit conversion:

$$t_{\text{sample}} > (R_S + R_I) \times \ln(2^{13}) \times C_I + 800\text{ns}$$

Substituting the values for  $R_I$  and  $C_I$  given above, the equation becomes:

$$t_{\text{sample}} > (R_S + 2\text{k}\Omega) \times 9.011 \times 40\text{pF} + 800\text{ns}$$

For example, if  $R_S$  is  $10\text{ k}\Omega$ ,  $t_{\text{sample}}$  must be greater than  $5.13\text{ }\mu\text{s}$ .

## 20.2.6 Conversion Memory

There are 16 ADC12MEMx conversion memory registers to store conversion results. Each ADC12MEMx is configured with an associated ADC12MCTLx control register. The SREFx bits define the voltage reference and the INCHx bits select the input channel. The EOS bit defines the end of sequence when a sequential conversion mode is used. A sequence rolls over from ADC12MEM15 to ADC12MEM0 when the EOS bit in ADC12MCTL15 is not set.

The CSTARTADDx bits define the first ADC12MCTLx used for any conversion. If the conversion mode is single-channel or repeat-single-channel the CSTARTADDx points to the single ADC12MCTLx to be used.

If the conversion mode selected is either sequence-of-channels or repeat-sequence-of-channels, CSTARTADDx points to the first ADC12MCTLx location to be used in a sequence. A pointer, not visible to software, is incremented automatically to the next ADC12MCTLx in a sequence when each conversion completes. The sequence continues until an EOS bit in ADC12MCTLx is processed - this is the last control byte processed.

When conversion results are written to a selected ADC12MEMx, the corresponding flag in the ADC12IFGx register is set.

## 20.2.7 ADC12 Conversion Modes

The ADC12 has four operating modes selected by the CONSEQx bits as discussed in Table 20–1.

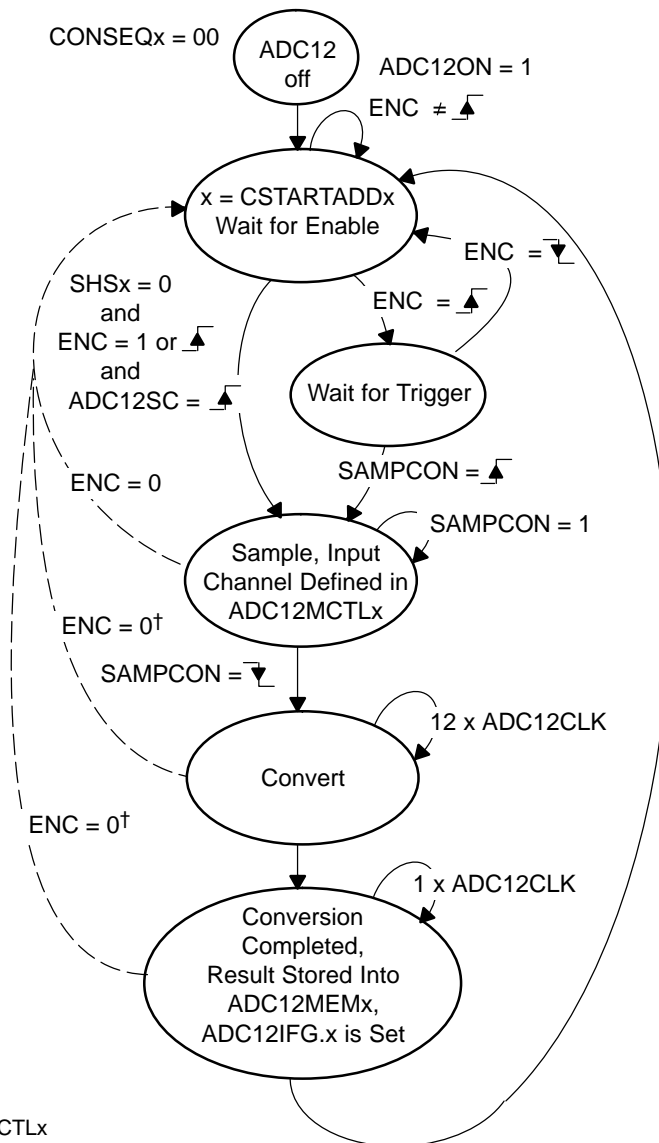
Table 20–1. Conversion Mode Summary

CONSEQx	Mode	Operation
00	Single channel single-conversion	A single channel is converted once.
01	Sequence-of-channels	A sequence of channels is converted once.
10	Repeat-single-channel	A single channel is converted repeatedly.
11	Repeat-sequence-of-channels	A sequence of channels is converted repeatedly.

### Single-Channel Single-Conversion Mode

A single channel is sampled and converted once. The ADC result is written to the ADC12MEMx defined by the CSTARTADDx bits. Figure 20–6 shows the flow of the Single-Channel, Single-Conversion mode. When ADC12SC triggers a conversion, successive conversions can be triggered by the ADC12SC bit. When any other trigger source is used, ENC must be toggled between each conversion.

Figure 20–6. Single-Channel, Single-Conversion Mode

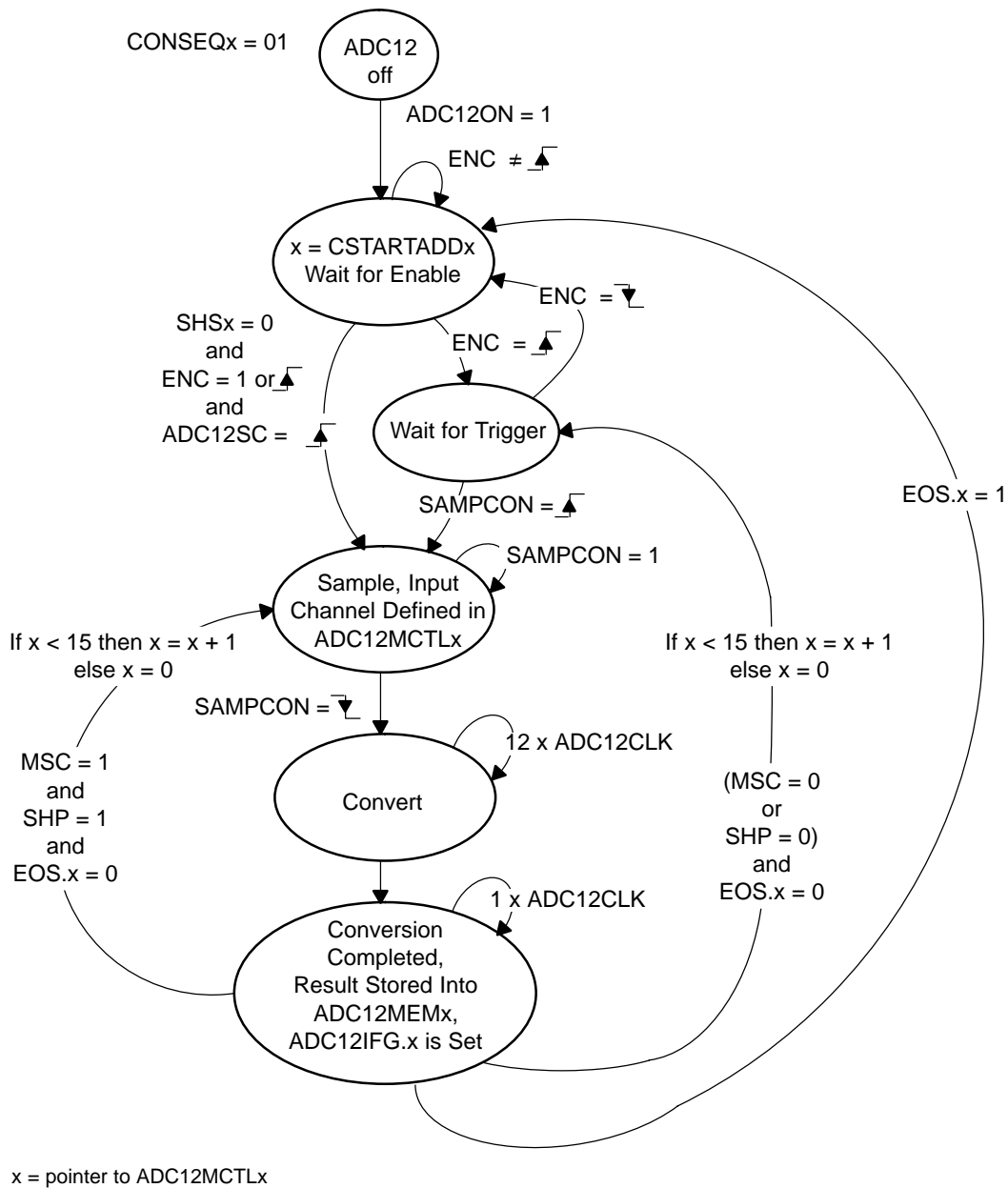


x = pointer to ADC12MCTLx  
 †Conversion result is unpredictable

### Sequence-of-Channels Mode

A sequence of channels is sampled and converted once. The ADC results are written to the conversion memories starting with the ADCMEMx defined by the CSTARTADDx bits. The sequence stops after the measurement of the channel with a set EOS bit. Figure 20–7 shows the sequence-of-channels mode. When ADC12SC triggers a sequence, successive sequences can be triggered by the ADC12SC bit. When any other trigger source is used, ENC must be toggled between each sequence.

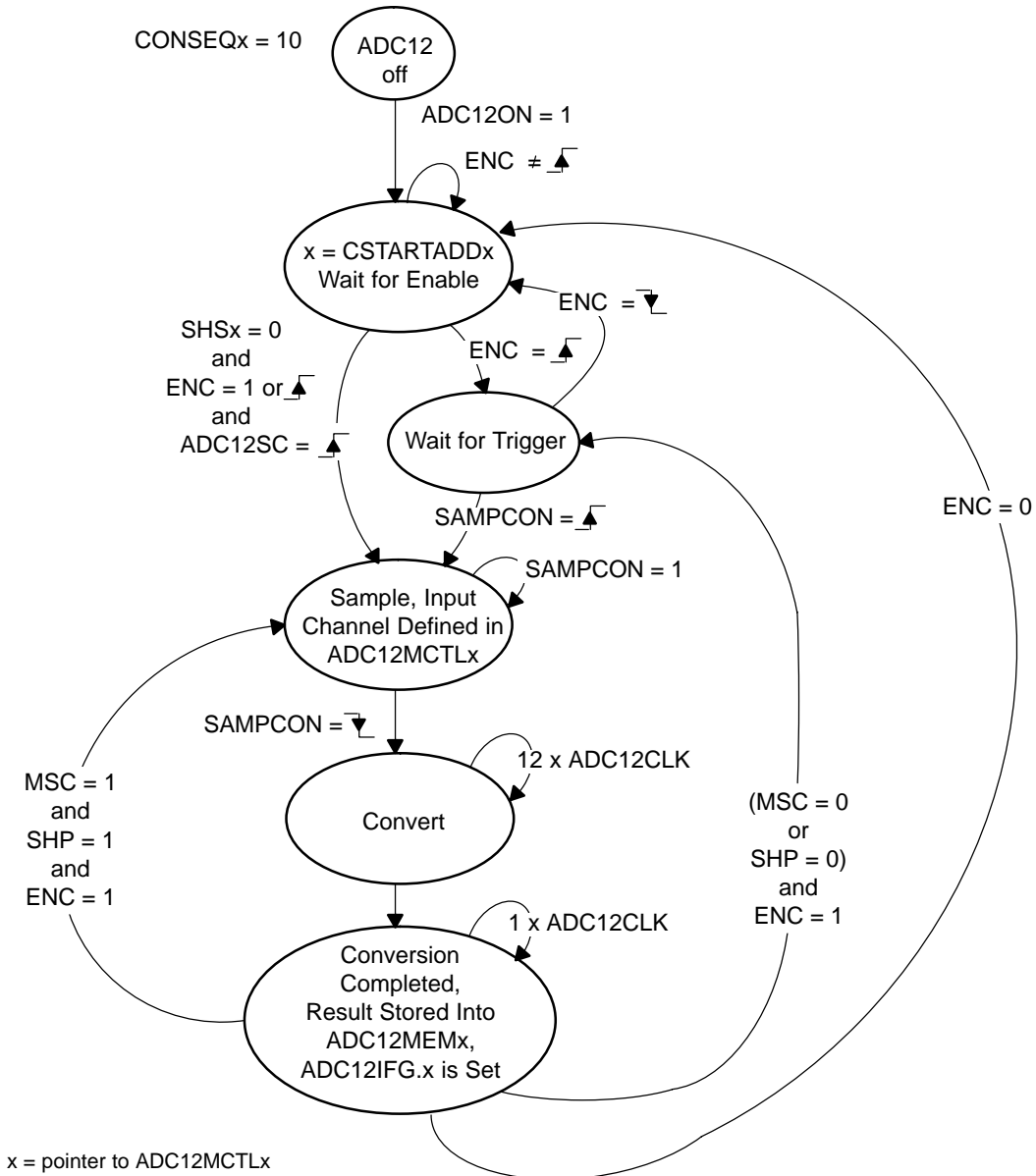
Figure 20–7. Sequence-of-Channels Mode



### Repeat-Single-Channel Mode

A single channel is sampled and converted continuously. The ADC results are written to the ADC12MEMx defined by the CSTARTADDx bits. It is necessary to read the result after the completed conversion because only one ADC12MEMx memory is used and is overwritten by the next conversion. Figure 20–8 shows repeat-single-channel mode

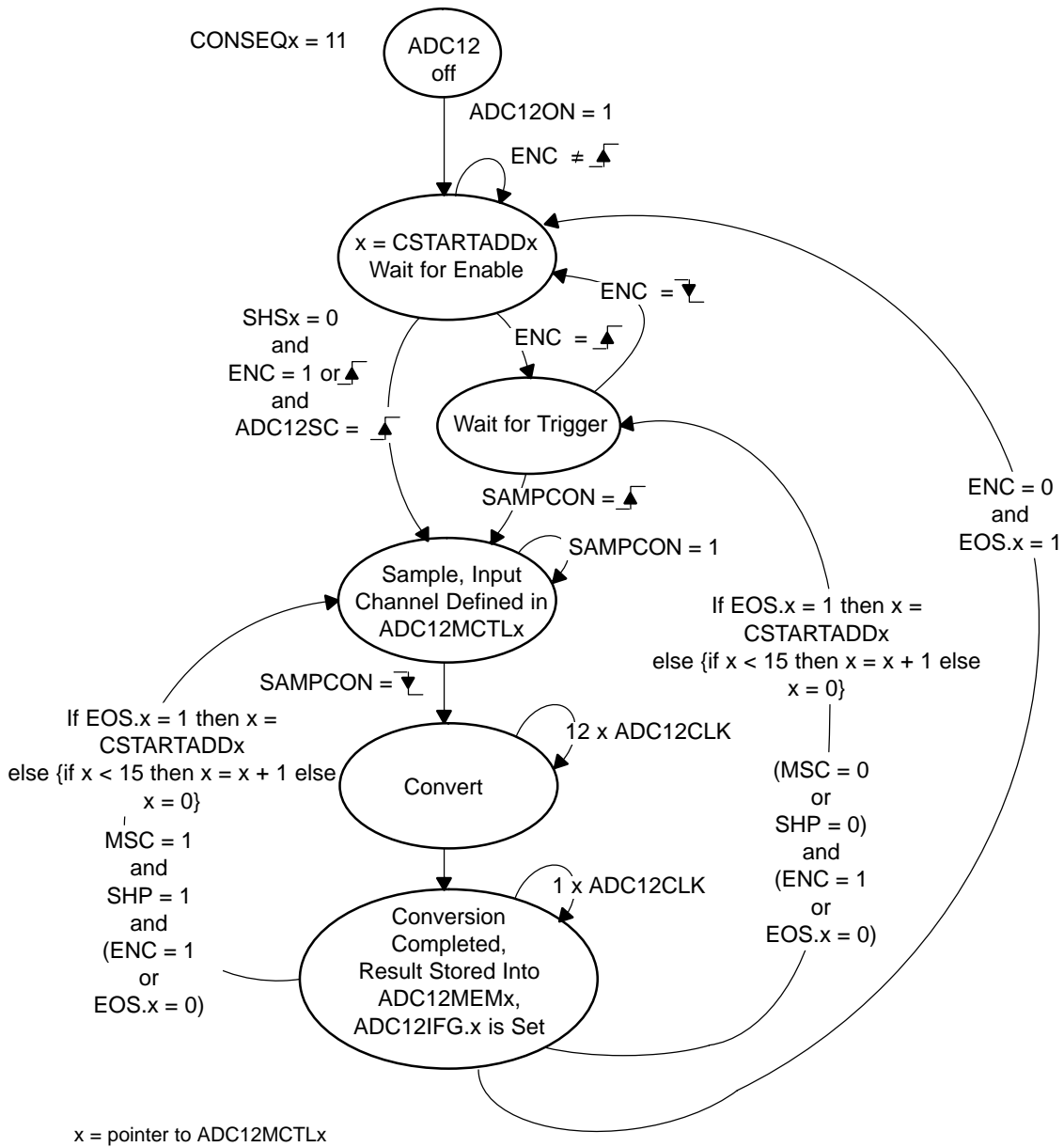
Figure 20–8. Repeat-Single-Channel Mode



### Repeat-Sequence-of-Channels Mode

A sequence of channels is sampled and converted repeatedly. The ADC results are written to the conversion memories starting with the ADC12MEMx defined by the CSTARTADDx bits. The sequence ends after the measurement of the channel with a set EOS bit and the next trigger signal re-starts the sequence. Figure 20–9 shows the repeat-sequence-of-channels mode.

Figure 20–9. Repeat-Sequence-of-Channels Mode



## Using the Multiple Sample and Convert (MSC) Bit

To configure the converter to perform successive conversions automatically and as quickly as possible, a multiple sample and convert function is available. When  $MSC = 1$ ,  $CONSEQx > 0$ , and the sample timer is used, the first rising edge of the SHI signal triggers the first conversion. Successive conversions are triggered automatically as soon as the prior conversion is completed. Additional rising edges on SHI are ignored until the sequence is completed in the single-sequence mode or until the ENC bit is toggled in repeat-single-channel, or repeated-sequence modes. The function of the ENC bit is unchanged when using the MSC bit.

## Stopping Conversions

Stopping ADC12 activity depends on the mode of operation. The recommended ways to stop an active conversion or conversion sequence are:

- Resetting ENC in single-channel single-conversion mode stops a conversion immediately and the results are unpredictable. For correct results, poll the busy bit until reset before clearing ENC.
- Resetting ENC during repeat-single-channel operation stops the converter at the end of the current conversion.
- Resetting ENC during a sequence or repeat-sequence mode stops the converter at the end of the sequence.
- Any conversion mode may be stopped immediately by setting the  $CONSEQx = 0$  and resetting ENC bit. Conversion data are unreliable.

---

**Note: No EOS Bit Set For Sequence**

If no EOS bit is set and a sequence mode is selected, resetting the ENC bit does not stop the sequence. To stop the sequence, first select a single-channel mode and then reset ENC.

---

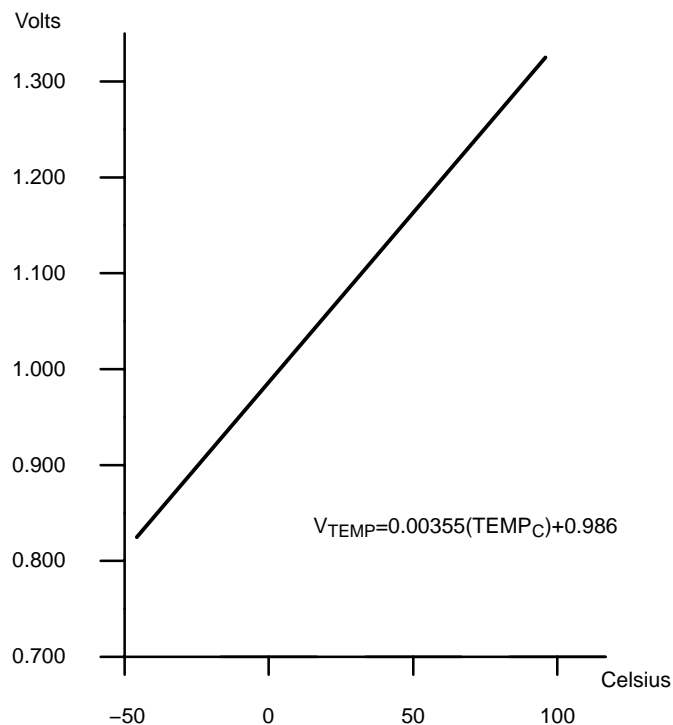
## 20.2.8 Using the Integrated Temperature Sensor

To use the on-chip temperature sensor, the user selects the analog input channel  $INCHx = 1010$ . Any other configuration is done as if an external channel was selected, including reference selection, conversion-memory selection, etc.

The typical temperature sensor transfer function is shown in Figure 20–10. When using the temperature sensor, the sample period must be greater than  $30\ \mu\text{s}$ . The temperature sensor offset error can be large, and may need to be calibrated for most applications. See device-specific datasheet for parameters.

Selecting the temperature sensor automatically turns on the on-chip reference generator as a voltage source for the temperature sensor. However, it does not enable the  $V_{REF+}$  output or affect the reference selections for the conversion. The reference choices for converting the temperature sensor are the same as with any other channel.

Figure 20–10. Typical Temperature Sensor Transfer Function





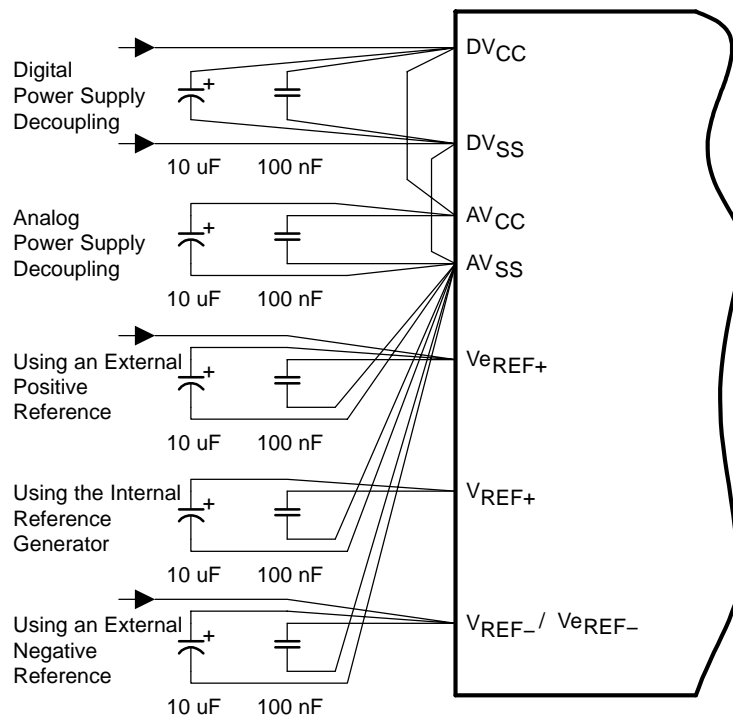
## 20.2.9 ADC12 Grounding and Noise Considerations

As with any high-resolution ADC, appropriate printed-circuit-board layout and grounding techniques should be followed to eliminate ground loops, unwanted parasitic effects, and noise.

Ground loops are formed when return current from the A/D flows through paths that are common with other analog or digital circuitry. If care is not taken, this current can generate small, unwanted offset voltages that can add to or subtract from the reference or input voltages of the A/D converter. The connections shown in Figure 20–11 help avoid this.

In addition to grounding, ripple and noise spikes on the power supply lines due to digital switching or switching power supplies can corrupt the conversion result. A noise-free design using separate analog and digital ground planes with a single-point connection is recommended to achieve high accuracy.

Figure 20–11. ADC12 Grounding and Noise Considerations



### 20.2.10 ADC12 Interrupts

The ADC12 has 18 interrupt sources:

- ADC12IFG0-ADC12IFG15
- ADC12OV, ADC12MEMx overflow
- ADC12TOV, ADC12 conversion time overflow

The ADC12IFGx bits are set when their corresponding ADC12MEMx memory register is loaded with a conversion result. An interrupt request is generated if the corresponding ADC12IE<sub>x</sub> bit and the GIE bit are set. The ADC12OV condition occurs when a conversion result is written to any ADC12MEMx before its previous conversion result was read. The ADC12TOV condition is generated when another sample-and-conversion is requested before the current conversion is completed.

### ADC12IV, Interrupt Vector Generator

All ADC12 interrupt sources are prioritized and combined to source a single interrupt vector. The interrupt vector register ADC12IV is used to determine which enabled ADC12 interrupt source requested an interrupt.

The highest priority enabled ADC12 interrupt generates a number in the ADC12IV register (see register description). This number can be evaluated or added to the program counter to automatically enter the appropriate software routine. Disabled ADC12 interrupts do not affect the ADC12IV value.

Any access, read or write, of the ADC12IV register automatically resets the ADC12OV condition or the ADC12TOV condition if either was the highest pending interrupt. Neither interrupt condition has an accessible interrupt flag. The ADC12IFGx flags are not reset by an ADC12IV access. ADC12IFGx bits are reset automatically by accessing their associated ADC12MEMx register or may be reset with software.

If another interrupt is pending after servicing of an interrupt, another interrupt is generated. For example, if the ADC12OV and ADC12IFG3 interrupts are pending when the interrupt service routine accesses the ADC12IV register, the ADC12OV interrupt condition is reset automatically. After the RETI instruction of the interrupt service routine is executed, the ADC12IFG3 generates another interrupt.

## ADC12 Interrupt Handling Software Example

The following software example shows the recommended use of ADC12IV and the handling overhead. The ADC12IV value is added to the PC to automatically jump to the appropriate routine.

The numbers at the right margin show the necessary CPU cycles for each instruction. The software overhead for different interrupt sources includes interrupt latency and return-from-interrupt cycles, but not the task handling itself. The latencies are:

- ADC12IFG0 - ADC12IFG14, ADC12TOV and ADC12OV    16 cycles
- ADC12IFG15    14 cycles

The interrupt handler for ADC12IFG15 shows a way to check immediately if a higher prioritized interrupt occurred during the processing of ADC12IFG15. This saves nine cycles if another ADC12 interrupt is pending.

```

; Interrupt handler for ADC12.
INT_ADC12          ; Enter Interrupt Service Routine          6
  ADD    &ADC12IV,PC; Add offset to PC                       3
  RETI    ; Vector 0: No interrupt                            5
  JMP    ADOV      ; Vector 2: ADC overflow                   2
  JMP    ADTOV    ; Vector 4: ADC timing overflow            2
  JMP    ADM0     ; Vector 6: ADC12IFG0                      2
  ...      ; Vectors 8-32                                    2
  JMP    ADM14    ; Vector 34: ADC12IFG14                   2
;
; Handler for ADC12IFG15 starts here. No JMP required.
;
ADM15    MOV    &ADC12MEM15,xxx; Move result, flag is reset
  ...      ; Other instruction needed?
  JMP    INT_ADC12 ; Check other int pending
;
; ADC12IFG14-ADC12IFG1 handlers go here
;
ADM0     MOV    &ADC12MEM0,xxx ; Move result, flag is reset
  ...      ; Other instruction needed?
  RETI    ; Return                                           5
;
ADTOV   ...      ; Handle Conv. time overflow
  RETI    ; Return                                           5
;
ADOV    ...      ; Handle ADCMEMx overflow
  RETI    ; Return                                           5

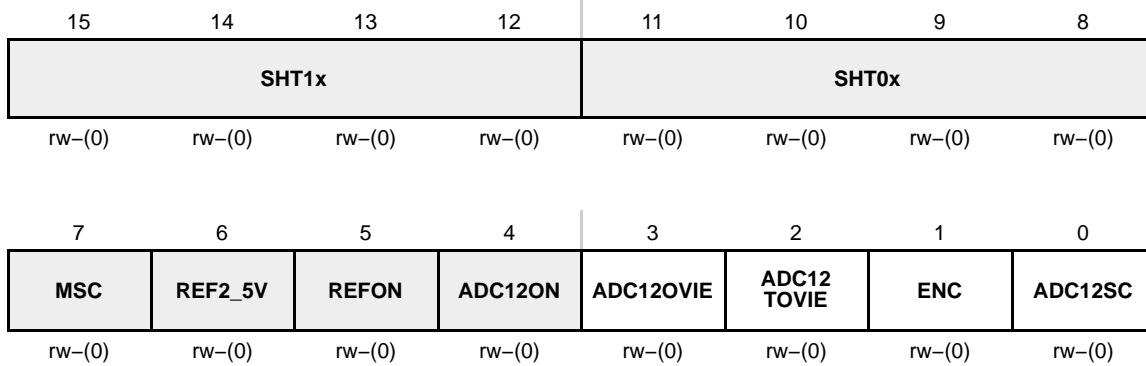
```

## 20.3 ADC12 Registers

The ADC12 registers are listed in Table 20–2 .

Table 20–2. ADC12 Registers

Register	Short Form	Register Type	Address	Initial State
ADC12 control register 0	ADC12CTL0	Read/write	01A0h	Reset with POR
ADC12 control register 1	ADC12CTL1	Read/write	01A2h	Reset with POR
ADC12 interrupt flag register	ADC12IFG	Read/write	01A4h	Reset with POR
ADC12 interrupt enable register	ADC12IE	Read/write	01A6h	Reset with POR
ADC12 interrupt vector word	ADC12IV	Read	01A8h	Reset with POR
ADC12 memory 0	ADC12MEM0	Read/write	0140h	Unchanged
ADC12 memory 1	ADC12MEM1	Read/write	0142h	Unchanged
ADC12 memory 2	ADC12MEM2	Read/write	0144h	Unchanged
ADC12 memory 3	ADC12MEM3	Read/write	0146h	Unchanged
ADC12 memory 4	ADC12MEM4	Read/write	0148h	Unchanged
ADC12 memory 5	ADC12MEM5	Read/write	014Ah	Unchanged
ADC12 memory 6	ADC12MEM6	Read/write	014Ch	Unchanged
ADC12 memory 7	ADC12MEM7	Read/write	014Eh	Unchanged
ADC12 memory 8	ADC12MEM8	Read/write	0150h	Unchanged
ADC12 memory 9	ADC12MEM9	Read/write	0152h	Unchanged
ADC12 memory 10	ADC12MEM10	Read/write	0154h	Unchanged
ADC12 memory 11	ADC12MEM11	Read/write	0156h	Unchanged
ADC12 memory 12	ADC12MEM12	Read/write	0158h	Unchanged
ADC12 memory 13	ADC12MEM13	Read/write	015Ah	Unchanged
ADC12 memory 14	ADC12MEM14	Read/write	015Ch	Unchanged
ADC12 memory 15	ADC12MEM15	Read/write	015Eh	Unchanged
ADC12 memory control 0	ADC12MCTL0	Read/write	080h	Reset with POR
ADC12 memory control 1	ADC12MCTL1	Read/write	081h	Reset with POR
ADC12 memory control 2	ADC12MCTL2	Read/write	082h	Reset with POR
ADC12 memory control 3	ADC12MCTL3	Read/write	083h	Reset with POR
ADC12 memory control 4	ADC12MCTL4	Read/write	084h	Reset with POR
ADC12 memory control 5	ADC12MCTL5	Read/write	085h	Reset with POR
ADC12 memory control 6	ADC12MCTL6	Read/write	086h	Reset with POR
ADC12 memory control 7	ADC12MCTL7	Read/write	087h	Reset with POR
ADC12 memory control 8	ADC12MCTL8	Read/write	088h	Reset with POR
ADC12 memory control 9	ADC12MCTL9	Read/write	089h	Reset with POR
ADC12 memory control 10	ADC12MCTL10	Read/write	08Ah	Reset with POR
ADC12 memory control 11	ADC12MCTL11	Read/write	08Bh	Reset with POR
ADC12 memory control 12	ADC12MCTL12	Read/write	08Ch	Reset with POR
ADC12 memory control 13	ADC12MCTL13	Read/write	08Dh	Reset with POR
ADC12 memory control 14	ADC12MCTL14	Read/write	08Eh	Reset with POR
ADC12 memory control 15	ADC12MCTL15	Read/write	08Fh	Reset with POR

**ADC12CTL0, ADC12 Control Register 0**

Modifiable only when ENC = 0

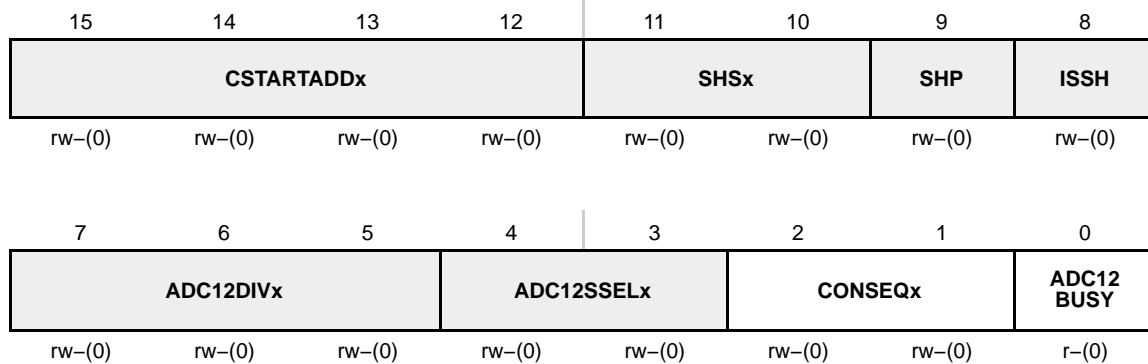
- SHT1x**      Bits      Sample-and-hold time. These bits define the number of ADC12CLK cycles in the sampling period for registers ADC12MEM8 to ADC12MEM15.  
15-12
- SHT0x**      Bits      Sample-and-hold time. These bits define the number of ADC12CLK cycles in the sampling period for registers ADC12MEM0 to ADC12MEM7.  
11-8

SHTx Bits	ADC12CLK cycles
0000	4
0001	8
0010	16
0011	32
0100	64
0101	96
0110	128
0111	192
1000	256
1001	384
1010	512
1011	768
1100	1024
1101	1024
1110	1024
1111	1024

---

<b>MSC</b>	Bit 7	Multiple sample and conversion. Valid only for sequence or repeated modes. 0 The sampling timer requires a rising edge of the SHI signal to trigger each sample-and-conversion. 1 The first rising edge of the SHI signal triggers the sampling timer, but further sample-and-conversions are performed automatically as soon as the prior conversion is completed.
<b>REF2_5V</b>	Bit 6	Reference generator voltage. REFON must also be set. 0 1.5 V 1 2.5 V
<b>REFON</b>	Bit 5	Reference generator on 0 Reference off 1 Reference on
<b>ADC12ON</b>	Bit 4	ADC12 on 0 ADC12 off 1 ADC12 on
<b>ADC12OVIE</b>	Bit 3	ADC12MEMx overflow-interrupt enable. The GIE bit must also be set to enable the interrupt. 0 Overflow interrupt disabled 1 Overflow interrupt enabled
<b>ADC12 TOVIE</b>	Bit 2	ADC12 conversion-time-overflow interrupt enable. The GIE bit must also be set to enable the interrupt. 0 Conversion time overflow interrupt disabled 1 Conversion time overflow interrupt enabled
<b>ENC</b>	Bit 1	Enable conversion 0 ADC12 disabled 1 ADC12 enabled
<b>ADC12SC</b>	Bit 0	Start conversion. Software-controlled sample-and-conversion start. ADC12SC and ENC may be set together with one instruction. ADC12SC is reset automatically. 0 No sample-and-conversion-start 1 Start sample-and-conversion

## ADC12CTL1, ADC12 Control Register 1

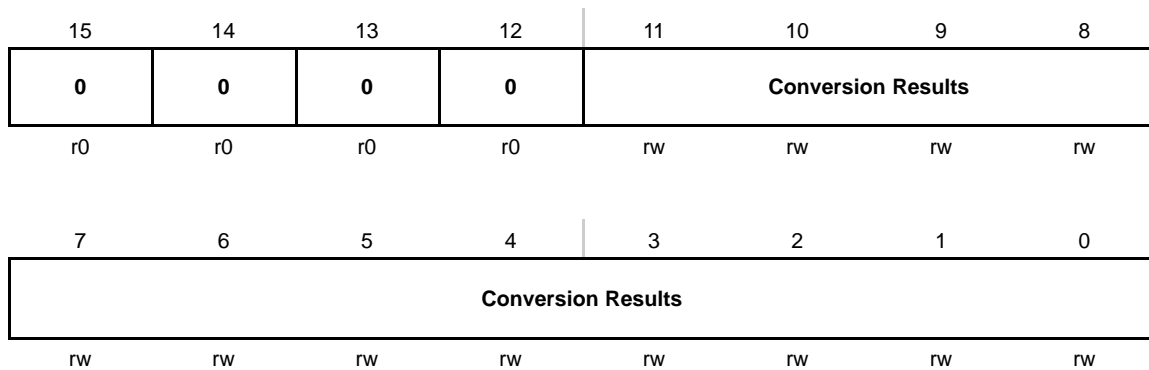


Modifiable only when ENC = 0

<b>CSTART ADDx</b>	Bits 15-12	Conversion start address. These bits select which ADC12 conversion-memory register is used for a single conversion or for the first conversion in a sequence. The value of CSTARTADDx is 0 to 0Fh, corresponding to ADC12MEM0 to ADC12MEM15.
<b>SHSx</b>	Bits 11-10	Sample-and-hold source select 00 ADC12SC bit 01 Timer_A.OUT1 10 Timer_B.OUT0 11 Timer_B.OUT1
<b>SHP</b>	Bit 9	Sample-and-hold pulse-mode select. This bit selects the source of the sampling signal (SAMPCON) to be either the output of the sampling timer or the sample-input signal directly. 0 SAMPCON signal is sourced from the sample-input signal. 1 SAMPCON signal is sourced from the sampling timer.
<b>ISSH</b>	Bit 8	Invert signal sample-and-hold 0 The sample-input signal is not inverted. 1 The sample-input signal is inverted.
<b>ADC12DIVx</b>	Bits 7-5	ADC12 clock divider 000 /1 001 /2 010 /3 011 /4 100 /5 101 /6 110 /7 111 /8

<b>ADC12 SSELx</b>	Bits 4-3	ADC12 clock source select 00 ADC12OSC 01 ACLK 10 MCLK 11 SMCLK
<b>CONSEQx</b>	Bits 2-1	Conversion sequence mode select 00 Single-channel, single-conversion 01 Sequence-of-channels 10 Repeat-single-channel 11 Repeat-sequence-of-channels
<b>ADC12 BUSY</b>	Bit 0	ADC12 busy. This bit indicates an active sample or conversion operation. 0 No operation is active. 1 A sequence, sample, or conversion is active.

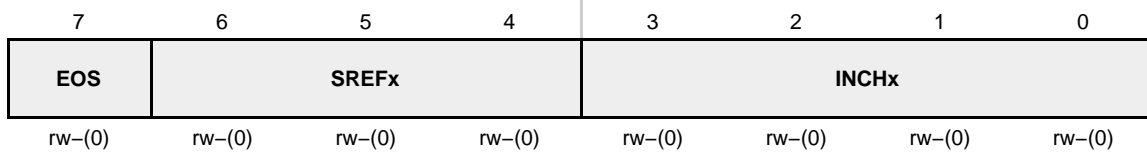
**ADC12MEMx, ADC12 Conversion Memory Registers**



**Conversion Results** Bits 15-0 The 12-bit conversion results are right-justified. Bit 11 is the MSB. Bits 15-12 are always 0. Writing to the conversion memory registers will corrupt the results.



## ADC12MCTLx, ADC12 Conversion Memory Control Registers



Modifiable only when ENC = 0

<b>EOS</b>	Bit 7	End of sequence. Indicates the last conversion in a sequence. 0 Not end of sequence 1 End of sequence
<b>SREFx</b>	Bits 6-4	Select reference 000 $V_{R+} = AV_{CC}$ and $V_{R-} = AV_{SS}$ 001 $V_{R+} = V_{REF+}$ and $V_{R-} = AV_{SS}$ 010 $V_{R+} = Ve_{REF+}$ and $V_{R-} = AV_{SS}$ 011 $V_{R+} = Ve_{REF+}$ and $V_{R-} = AV_{SS}$ 100 $V_{R+} = AV_{CC}$ and $V_{R-} = V_{REF-} / Ve_{REF-}$ 101 $V_{R+} = V_{REF+}$ and $V_{R-} = V_{REF-} / Ve_{REF-}$ 110 $V_{R+} = Ve_{REF+}$ and $V_{R-} = V_{REF-} / Ve_{REF-}$ 111 $V_{R+} = Ve_{REF+}$ and $V_{R-} = V_{REF-} / Ve_{REF-}$
<b>INCHx</b>	Bits 3-0	Input channel select 0000 A0 0001 A1 0010 A2 0011 A3 0100 A4 0101 A5 0110 A6 0111 A7 1000 $V_{REF+}$ 1001 $V_{REF-} / Ve_{REF-}$ 1010 Temperature sensor 1011 $(AV_{CC} - AV_{SS}) / 2$ 1100 $(AV_{CC} - AV_{SS}) / 2$ , A12 on MSP430FG43x devices 1101 $(AV_{CC} - AV_{SS}) / 2$ , A13 on MSP430FG43x devices 1110 $(AV_{CC} - AV_{SS}) / 2$ , A14 on MSP430FG43x devices 1111 $(AV_{CC} - AV_{SS}) / 2$ , A15 on MSP430FG43x devices

**ADC12IE, ADC12 Interrupt Enable Register**

15	14	13	12	11	10	9	8
<b>ADC12IE15</b>	<b>ADC12IE14</b>	<b>ADC12IE13</b>	<b>ADC12IE12</b>	<b>ADC12IE11</b>	<b>ADC12IE10</b>	<b>ADC12IE9</b>	<b>ADC12IE8</b>
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
<b>ADC12IE7</b>	<b>ADC12IE6</b>	<b>ADC12IE5</b>	<b>ADC12IE4</b>	<b>ADC12IE3</b>	<b>ADC12IE2</b>	<b>ADC12IE1</b>	<b>ADC12IE0</b>
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

**ADC12IE<sub>x</sub>** Bits 15-0 Interrupt enable. These bits enable or disable the interrupt request for the ADC12IFG<sub>x</sub> bits.  
 0 Interrupt disabled  
 1 Interrupt enabled

**ADC12IFG, ADC12 Interrupt Flag Register**

15	14	13	12	11	10	9	8
<b>ADC12IFG15</b>	<b>ADC12IFG14</b>	<b>ADC12IFG13</b>	<b>ADC12IFG12</b>	<b>ADC12IFG11</b>	<b>ADC12IFG10</b>	<b>ADC12IFG9</b>	<b>ADC12IFG8</b>
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
<b>ADC12IFG7</b>	<b>ADC12IFG6</b>	<b>ADC12IFG5</b>	<b>ADC12IFG4</b>	<b>ADC12IFG3</b>	<b>ADC12IFG2</b>	<b>ADC12IFG1</b>	<b>ADC12IFG0</b>
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

**ADC12IFG<sub>x</sub>** Bits 15-0 ADC12MEM<sub>x</sub> Interrupt flag. These bits are set when corresponding ADC12MEM<sub>x</sub> is loaded with a conversion result. The ADC12IFG<sub>x</sub> bits are reset if the corresponding ADC12MEM<sub>x</sub> is accessed, or may be reset with software.  
 0 No interrupt pending  
 1 Interrupt pending

**ADC12IV, ADC12 Interrupt Vector Register**

**ADC12IVx**    Bits    ADC12 interrupt vector value  
                  15-0

ADC12IV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
000h	No interrupt pending	–	
002h	ADC12MEMx overflow	–	Highest
004h	Conversion time overflow	–	
006h	ADC12MEM0 interrupt flag	ADC12IFG0	
008h	ADC12MEM1 interrupt flag	ADC12IFG1	
00Ah	ADC12MEM2 interrupt flag	ADC12IFG2	
00Ch	ADC12MEM3 interrupt flag	ADC12IFG3	
00Eh	ADC12MEM4 interrupt flag	ADC12IFG4	
010h	ADC12MEM5 interrupt flag	ADC12IFG5	
012h	ADC12MEM6 interrupt flag	ADC12IFG6	
014h	ADC12MEM7 interrupt flag	ADC12IFG7	
016h	ADC12MEM8 interrupt flag	ADC12IFG8	
018h	ADC12MEM9 interrupt flag	ADC12IFG9	
01Ah	ADC12MEM10 interrupt flag	ADC12IFG10	
01Ch	ADC12MEM11 interrupt flag	ADC12IFG11	
01Eh	ADC12MEM12 interrupt flag	ADC12IFG12	
020h	ADC12MEM13 interrupt flag	ADC12IFG13	
022h	ADC12MEM14 interrupt flag	ADC12IFG14	
024h	ADC12MEM15 interrupt flag	ADC12IFG15	Lowest

**SD16**

---

---

---

---

The SD16 module is a multichannel 16-bit, sigma-delta analog-to-digital converter. This chapter describes the SD16. The SD16 module is implemented in the MSP430FE42x and MSP430F42x devices.

<b>Topic</b>	<b>Page</b>
<b>21.1 SD16 Introduction .....</b>	<b>21-2</b>
<b>21.2 SD16 Operation .....</b>	<b>21-4</b>
<b>21.3 SD16 Registers .....</b>	<b>21-18</b>

## 21.1 SD16 Introduction

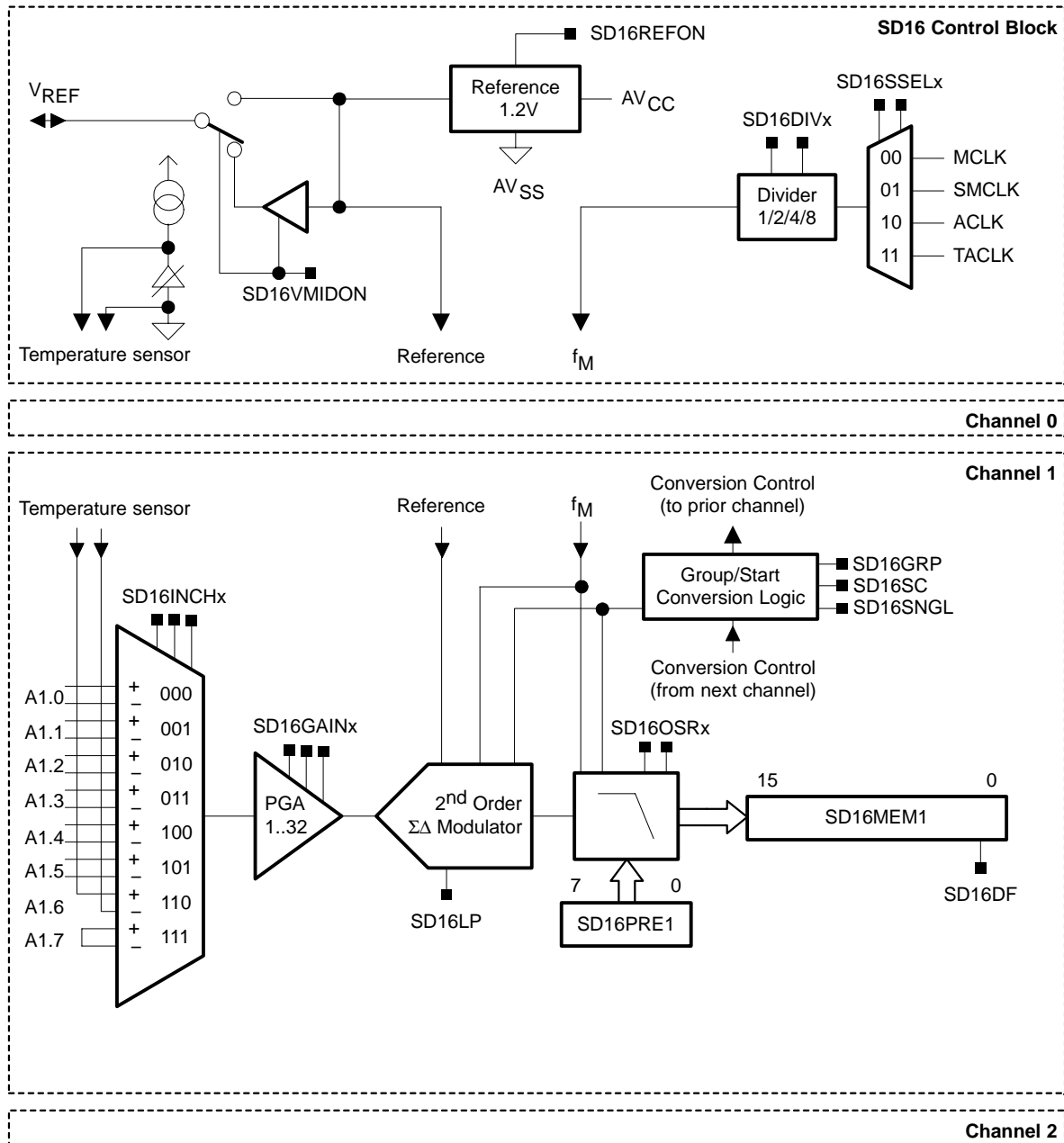
The SD16 module consists of up to three independent sigma-delta analog-to-digital converters and an internal voltage reference. Each channel has up to 8 fully differential multiplexed inputs including a built-in temperature sensor. The converters are based on second-order oversampling sigma-delta modulators and digital decimation filters. The decimation filters are comb type filters with selectable oversampling ratios of up to 256. Additional filtering can be done in software.

Features of the SD16 include:

- 16-bit sigma-delta architecture
- Up to 3 independent, simultaneously-sampling ADC channels
- Up to 8 multiplexed differential analog inputs per channel
- Software selectable on-chip reference voltage generation (1.2V)
- Software selectable internal or external reference
- Built-in temperature sensor accessible by all channels
- Up to 1.048576 MHz modulator input frequency
- Selectable low-power conversion mode

The block diagram of the SD16 module is shown in Figure 21–1.

Figure 21–1. SD16 Block Diagram



## 21.2 SD16 Operation

The SD16 module is configured with user software. The setup and operation of the SD16 is discussed in the following sections.

### 21.2.1 ADC Core

The analog-to-digital conversion is performed by a 1-bit, second-order sigma-delta modulator. A single-bit comparator within the modulator quantizes the input signal with the modulator frequency  $f_M$ . The resulting 1-bit data stream is averaged by the digital filter for the conversion result.

### 21.2.2 Analog Input Range and PGA

The full-scale input voltage range for each analog input pair is dependent on the gain setting of the programmable gain amplifier of each channel. The maximum full-scale range is  $\pm V_{FSR}$  where  $V_{FSR}$  is defined by:

$$V_{FSR} = \frac{V_{REF}/2}{GAIN_{PGA}}$$

For a 1.2V reference, the maximum full-scale input range for a gain of 1 is:

$$\pm V_{FSR} = \frac{1.2V/2}{1} = \pm 0.6V$$

Refer to the device-specific data sheet for full-scale input specifications.

### 21.2.3 Voltage Reference Generator

The SD16 module has a built-in 1.2V reference that can be used for each SD16 channel and is enabled by the SD16REFON bit. When using the internal reference an external 100nF capacitor connected from  $V_{REF}$  to  $AV_{SS}$  is recommended to reduce noise. The internal reference voltage can be used off-chip when SD16VMIDON = 1. The buffered output can provide up to 1mA of drive. When using the internal reference off-chip, a 470nF capacitor connected from  $V_{REF}$  to  $AV_{SS}$  is required. See device-specific data sheet for parameters.

An external voltage reference can be applied to the  $V_{REF}$  input when SD16REFON and SD16VMIDON are both reset.

### 21.2.4 Auto Power-Down

The SD16 is designed for low power applications. When the SD16 is not actively converting, it is automatically disabled and automatically re-enabled when needed. The reference is not automatically disabled, but can be disabled by setting SD16REFON = 0. When the SD16 or reference are disabled, they consume no current.

### 21.2.5 Channel Selection

Each SD16 channel can convert up to 8 differential pair inputs multiplexed into the PGA. Up to six input pairs (A0-A5) are available externally on the device. See the device-specific data sheet for analog input pin information. An internal temperature sensor is available to each channel using the A6 multiplexer input. Input A7 is a shorted connection between the + and - input pair and can be used to calibrate the offset of each SD16 input stage.

### Analog Input Setup

The analog input of each channel is configured using the SD16INCTLx register. These settings can be independently configured for each SD16 channel.

The SD16INCHx bits select one of eight differential input pairs of the analog multiplexer. The gain for each PGA is selected by the SD16GAINx bits. A total of six gain settings are available.

During conversion any modification to the SD16INCHx and SD16GAINx bits will become effective with the next decimation step of the digital filter. After these bits are modified, the next three conversions may be invalid due to the settling time of the digital filter. This can be handled automatically with the SD16INTDLYx bits. When SD16INTDLY = 00h, conversion interrupt requests will not begin until the 4<sup>th</sup> conversion after a start condition.

### Analog Input Characteristics

The SD16 uses a switched-capacitor input stage that appears as an impedance to external circuitry. The equivalent impedance differs for the PGA settings and is given in the device-specific datasheet.

### Anti-Aliasing Filter

An external RC anti-aliasing filter is recommended for the SD16 to prevent aliasing of the input signal. The cutoff frequency should be < 10 kHz for a 1 Mhz modulator clock and OSR = 256. The cutoff frequency may set to a lower frequency for applications that have lower bandwidth requirements.



### 21.2.6 Digital Filter

The digital filter processes the 1-bit data stream from the modulator using a SINC<sup>3</sup> comb filter. The transfer function is described in the z-Domain by:

$$H(z) = \left( \frac{1}{OSR} \times \frac{1 - z^{-OSR}}{1 - z^{-1}} \right)^3$$

and in the frequency domain by:

$$H(f) = \left[ \frac{\text{sinc}\left(OSR\pi\frac{f}{f_M}\right)}{\text{sinc}\left(\pi\frac{f}{f_M}\right)} \right]^3 = \left[ \frac{1}{OSR} \times \frac{\sin\left(OSR \times \pi \times \frac{f}{f_M}\right)}{\sin\left(\pi \times \frac{f}{f_M}\right)} \right]^3$$

where the oversampling rate, OSR, is the ratio of the modulator frequency  $f_M$  to the sample frequency  $f_S$ . Figure 21–2 shows the filter's frequency response for an OSR of 32. The first filter notch is at  $f_S = f_M/OSR$ . The notch's frequency can be adjusted by changing the modulator's frequency,  $f_M$ , using SD16SSELx and SD16DIVx and the oversampling rate using SD16OSRx.

The digital filter for each enabled ADC channel completes the decimation of the digital bit-stream and outputs new conversion results to the corresponding SD16MEMx register at the sample frequency  $f_S$ .

Figure 21–2. Comb Filter's Frequency Response with OSR = 32

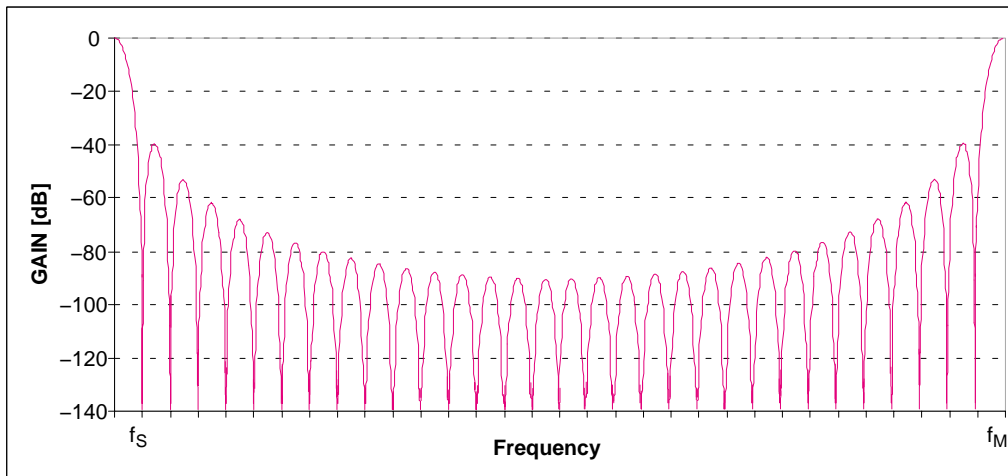
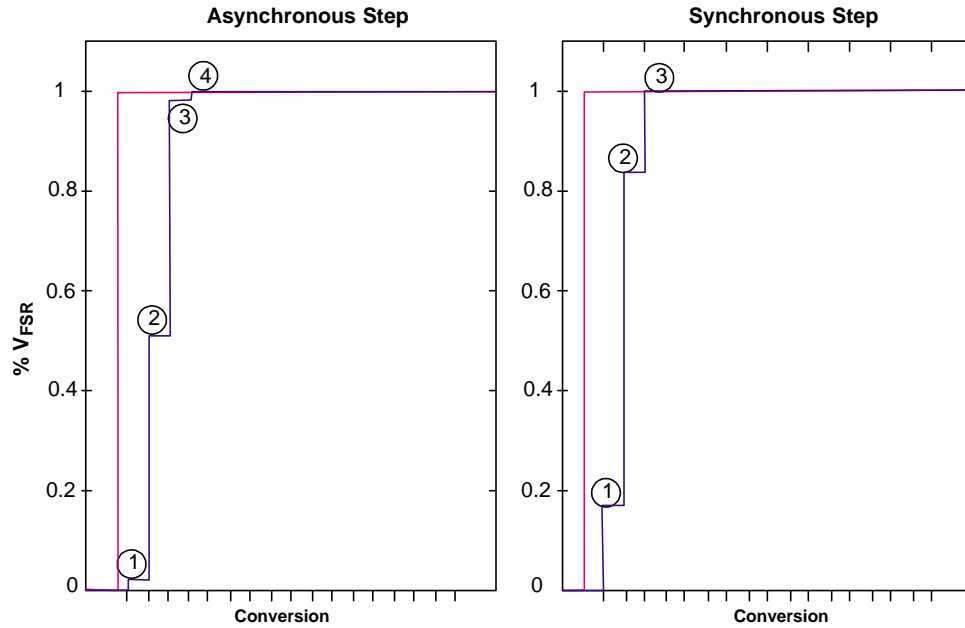


Figure 21–3 shows the digital filter step response and conversion points. For step changes at the input after start of conversion a settling time must be allowed before a valid conversion result is available. The SD16INTDLYx bits can provide sufficient filter settling time for a full-scale change at the ADC input. If the step occurs synchronously to the decimation of the digital filter the valid data will be available on the third conversion. An asynchronous step will require one additional conversion before valid data is available.

Figure 21–3. Digital Filter Step Response and Conversion Points



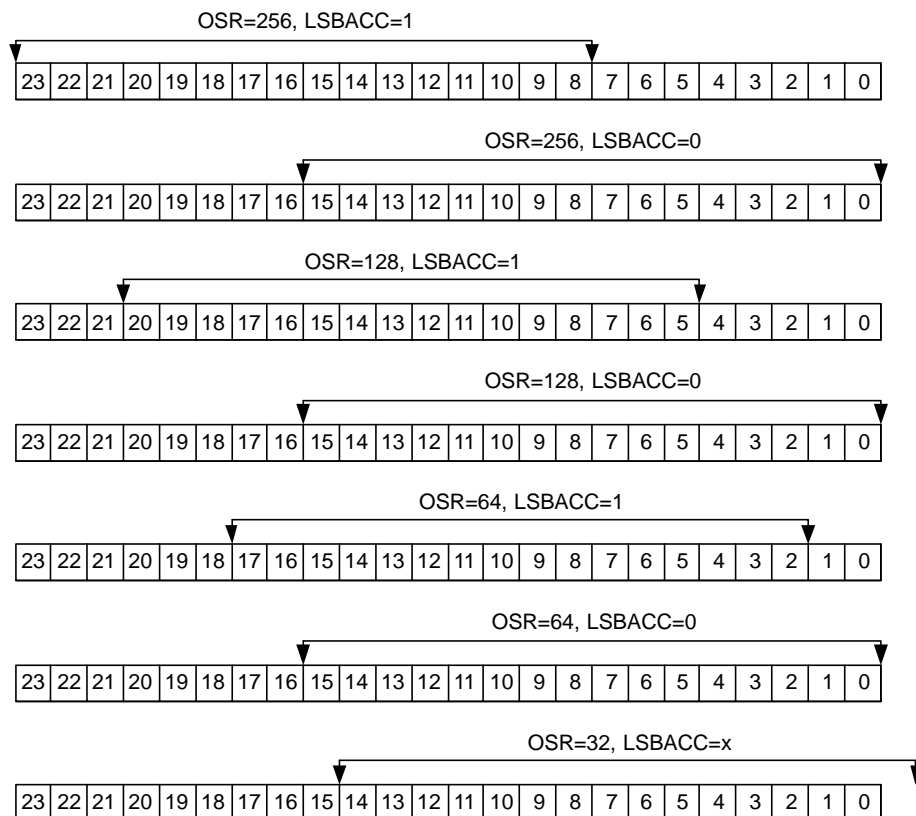
## Digital Filter Output

The number of bits output by each digital filter is dependent on the oversampling ratio and ranges from 16 to 24 bits. Figure 21–4 shows the digital filter output bits and their relation to SD16MEMx for each OSR. For example, for OSR = 256 and LSBACC = 1, the SD16MEMx register contains bits 23 – 8 of the digital filter output. When OSR = 32, the SD16MEMx LSB is always zero.

The SD16LSBACC and SD16LSBTOG bits give access to the least significant bits of the digital filter output. When SD16LSBACC = 1 the 16 least significant bits of the digital filter's output are read from SD16MEMx using word instructions. The SD16MEMx register can also be accessed with byte instructions returning only the 8 least significant bits of the digital filter output.

When SD16LSBTOG = 1 the SD16LSBACC bit is automatically toggled each time the corresponding channel's SD16MEMx register is read. This allows the complete digital filter output result to be read with two read accesses of SD16MEMx. Setting or clearing SD16LSBTOG does not change SD16LSBACC until the next SD16MEMx access.

Figure 21–4. Used Bits of Digital Filter Output.



## 21.2.7 Conversion Memory Registers: SD16MEMx

One SD16MEMx register is associated with each SD16 channel. Conversion results for each channel are moved to the corresponding SD16MEMx register with each decimation step of the digital filter. The SD16IFG bit for a given channel is set when new data is written to SD16MEMx. SD16IFG is automatically cleared when SD16MEMx is read by the CPU or may be cleared with software.

### Output Data Format

The output data format is configurable in two's complement or offset binary as shown in Table 21–1. The data format is selected by the SD16DF bit.

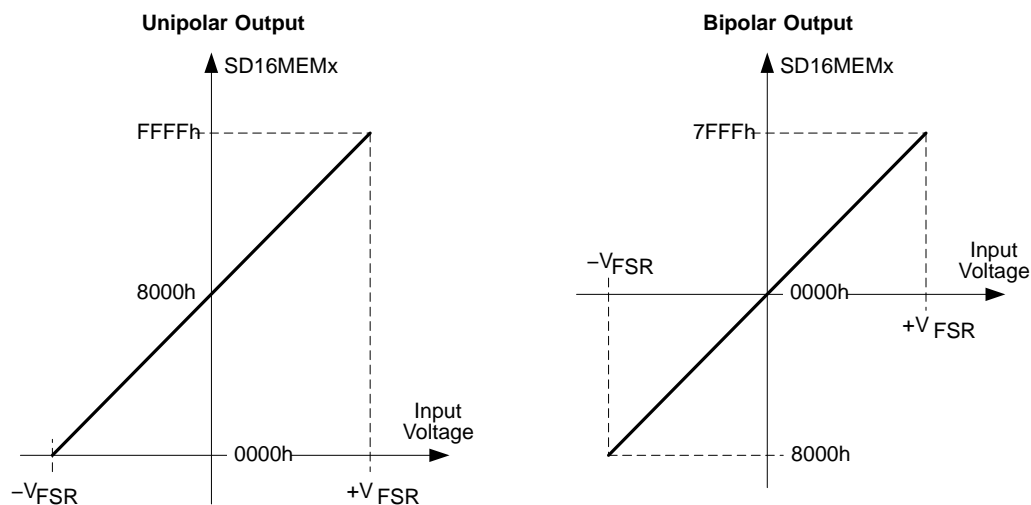
Table 21–1. Data Format

SD16DF	Format	Analog Input	SD16MEMx†	Digital Filter Output (OSR =256)
0	Unipolar: Offset Binary	+FSR	FFFF	FFFFFF
		ZERO	8000	800000
		–FSR	0000	000000
1	Bipolar: Two's complement	+FSR	7FFF	7FFFFFF
		ZERO	0000	000000
		–FSR	8000	800000

† Independent of SD16OSRx setting; SD16LSBACC = 0.

Figure 21–5 shows the relationship between the full-scale input voltage range from  $-V_{FSR}$  to  $+V_{FSR}$  and the conversion result. The digital values for both data formats are illustrated.

Figure 21–5. Input Voltage vs. Digital Output



### 21.2.8 Conversion Modes

The SD16 module can be configured for four modes of operation, listed in Table 21–2. The SD16SNGL and SD16GRP bits for each channel selects the conversion mode.

Table 21–2. Conversion Mode Summary

SD16SNGL	SD16GRP†	Mode	Operation
1	0	Single channel, Single conversion	A single channel is converted once.
0	0	Single channel, Continuous conversion	A single channel is converted continuously.
1	1	Group of channels, Single conversion	A group of channels is converted once.
0	1	Group of channels, Continuous conversion	A group of channels is converted continuously.

† A channel is grouped and is the master channel of the group when SD16GRP = 0 if SD16GRP for the prior channel(s) is set.

#### Single Channel, Single Conversion

Setting the SD16SC bit of a channel initiates one conversion on that channel when SD16SNGL = 1 and it is not grouped with any other channels. The SD16SC bit will automatically be cleared after conversion completion.

Clearing SD16SC before the conversion is completed immediately stops conversion of the selected channel, the channel is powered down and the corresponding digital filter is turned off. The value in SD16MEMx can change when SD16SC is cleared. It is recommended that the conversion data in SD16MEMx be read prior to clearing SD16SC to avoid reading an invalid result.

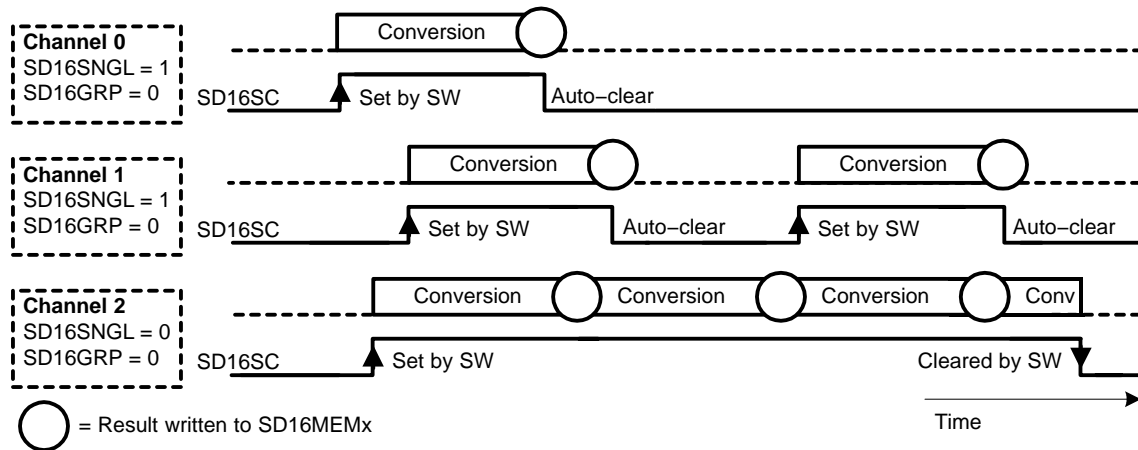
#### Single Channel, Continuous Conversion

When SD16SNGL = 0 continuous conversion mode is selected. Conversion of the selected channel will begin when SD16SC is set and continue until the SD16SC bit is cleared by software when the channel is not grouped with any other channel.

Clearing SD16SC immediately stops conversion of the selected channel, the channel is powered down and the corresponding digital filter is turned off. The value in SD16MEMx can change when SD16SC is cleared. It is recommended that the conversion data in SD16MEMx be read prior to clearing SD16SC to avoid reading an invalid result.

Figure 21–6 shows single channel operation for single conversion mode and continuous conversion mode.

Figure 21–6. Single Channel Operation



### Group of Channels, Single Conversion

Consecutive SD16 channels can be grouped together with the SD16GRP bit to synchronize conversions. Setting SD16GRP for a channel groups that channel with the next channel in the module. For example, setting SD16GRP for channel 0 groups that channel with channel 1. In this case, channel 1 is the master channel, enabling and disabling conversion of all channels in the group with its SD16SC bit. The SD16GRP bit of the master channel is always 0. The SD16GRP bit of last channel in SD16 has no function and is always 0.

When SD16SNGL = 1 for a channel in a group, single conversion mode is selected. A single conversion of that channel will occur synchronously when the master channel SD16SC bit is set. The SD16SC bit of all channels in the group will automatically be set and cleared by SD16SC of the master channel. SD16SC for each channel can also be cleared in software independently.

Clearing SD16SC of the master channel before the conversions are completed immediately stops conversions of all channels in the group, the channels are powered down and the corresponding digital filters are turned off. Values in SD16MEMx can change when SD16SC is cleared. It is recommended that the conversion data in SD16MEMx be read prior to clearing SD16SC to avoid reading an invalid result.

### Group of Channels, Continuous Conversion

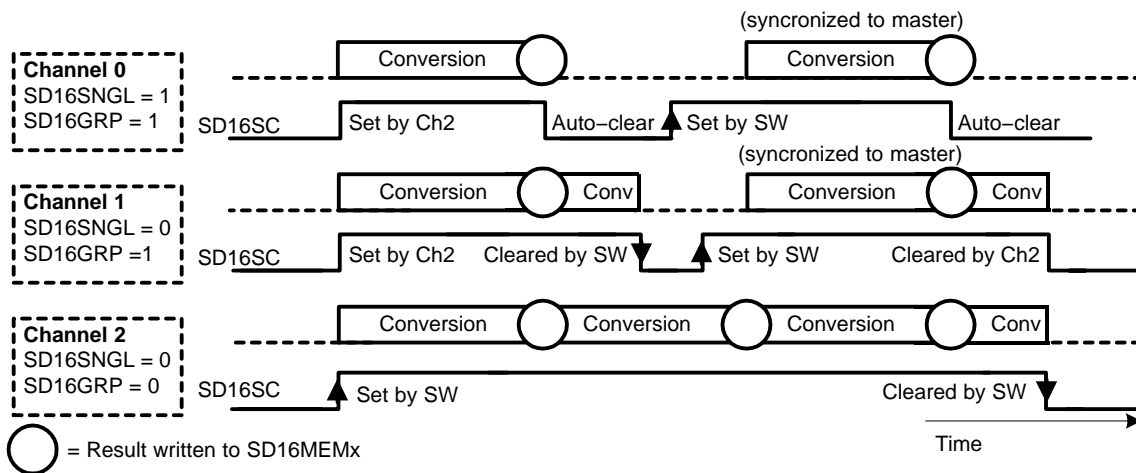
When SD16SNGL = 0 for a channel in a group, continuous conversion mode is selected. Continuous conversion of that channel will occur synchronously when the master channel SD16SC bit is set. SD16SC bits for all grouped channels will be automatically set and cleared with the master channel's SD16SC bit. SD16SC for each channel in the group can also be cleared in software independently.

When SD16SC of a grouped channel is set by software independently of the master, conversion of that channel will automatically synchronize to conversions of the master channel. This ensures that conversions for grouped channels are always synchronous to the master.

Clearing SD16SC of the master channel immediately stops conversions of all channels in the group the channels are powered down and the corresponding digital filters are turned off. Values in SD16MEMx can change when SD16SC is cleared. It is recommended that the conversion data in SD16MEMx be read prior to clearing SD16SC to avoid reading an invalid result.

Figure 21–7 shows grouped channel operation for three SD16 channels. Channel 0 is configured for single conversion mode, SD16SNGL = 1, and channels 1 and 2 are in continuous conversion mode, SD16SNGL = 0. Channel two, the last channel in the group, is the master channel. Conversions of all channels in the group occur synchronously to the master channel regardless of when each SD16SC bit is set using software.

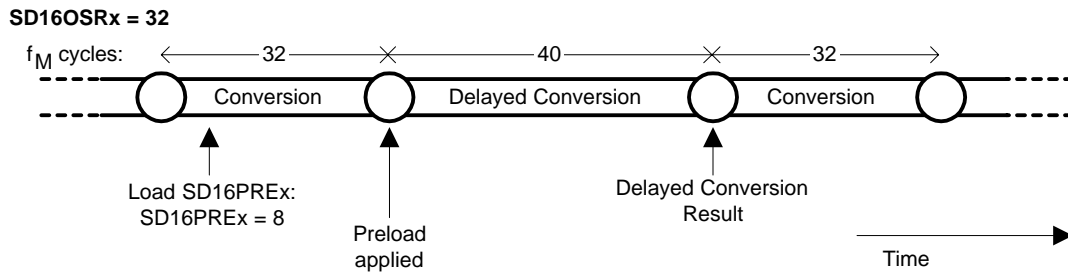
Figure 21–7. Grouped Channel Operation



### 21.2.9 Conversion Operation Using Preload

When multiple channels are grouped the SD16PREx registers can be used to delay the conversion time frame for each channel. Using SD16PREx, the decimation time of the digital filter is increased by the specified number of  $f_M$  clock cycles and can range from 0 to 255. Figure 21–8 shows an example using SD16PREx.

Figure 21–8. Conversion Delay using Preload



The SD16PREx delay is applied to the beginning of the next conversion cycle after being written. The delay is used on the first conversion after SD16SC is set and on the conversion cycle following each write to SD16PREx. Following conversions are not delayed. After modifying SD16PREx, the next write to SD16PREx should not occur until the next conversion cycle is completed, otherwise the conversion results may be incorrect.

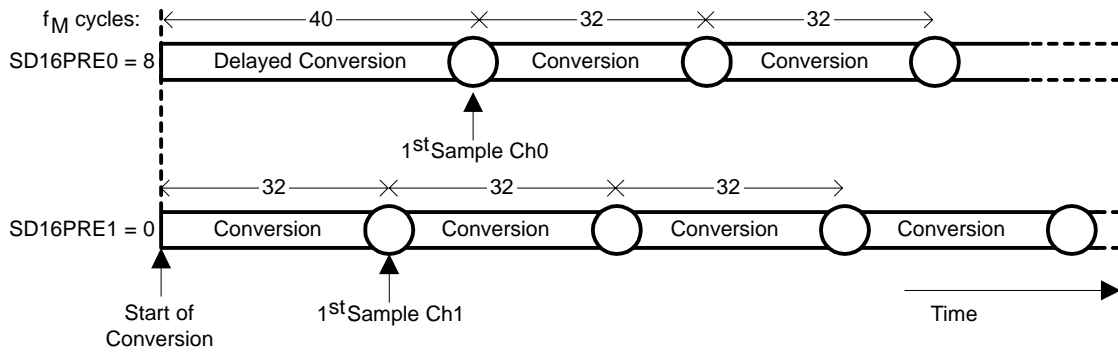
The accuracy of the result for the delayed conversion cycle using SD16PREx is dependent on the length of the delay and the frequency of the analog signal being sampled. For example, when measuring a DC signal, SD16PREx delay has no effect on the conversion result regardless of the duration. The user must determine when the delayed conversion result is useful in their application.

Figure 21–9 shows the operation of grouped channels 0 and 1. The preload register of channel 1 is loaded with zero resulting in immediate conversion whereas the conversion cycle of channel 0 is delayed by setting SD16PRE0 = 8. The first channel 0 conversion uses SD16PREx = 8, shifting all subsequent conversions by 8  $f_M$  clock cycles.



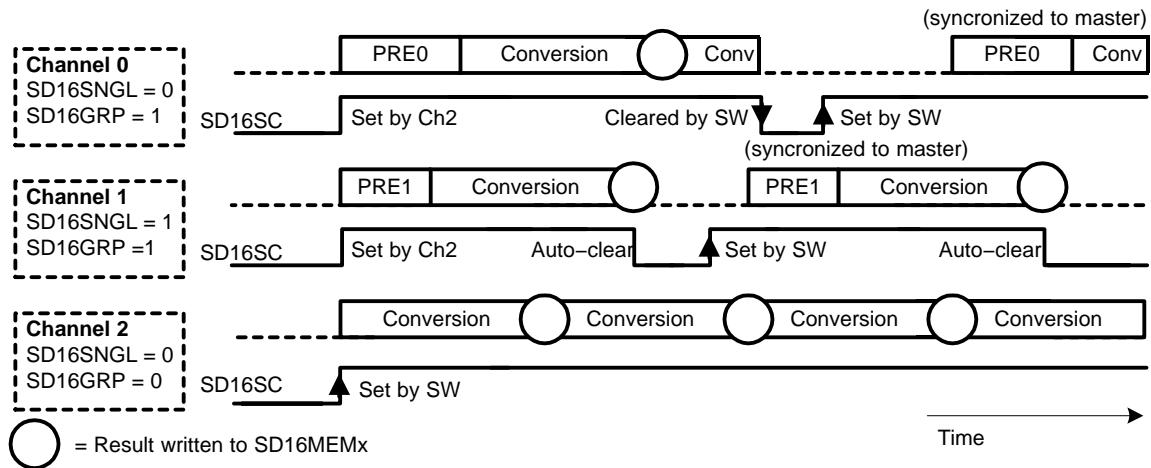
Figure 21–9. Start of Conversion using Preload

SD16OSRx = 32



When channels are grouped, care must be taken when a channel or channels operate in single conversion mode or are disabled in software while the master channel remains active. Each time channels in the group are re-enabled and re-synchronize with the master channel, the preload delay for that channel will be reintroduced. Figure 21–10 shows the re-synchronization and preload delays for channels in a group. It is recommended that SD16PREx = 0 for the master channel to maintain a consistent delay between the master and remaining channels in the group when they are re-enabled.

Figure 21–10. Preload and Channel Synchronization

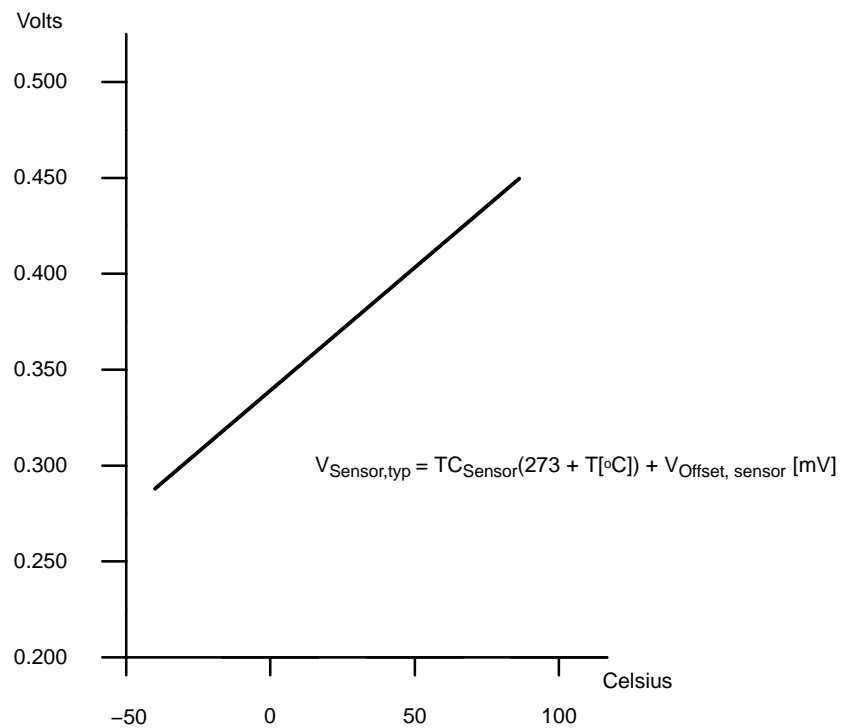


### 21.2.10 Using the Integrated Temperature Sensor

To use the on-chip temperature sensor, the user selects the analog input channel  $SD16INCHx = 110$ . Any other configuration is done as if an external channel was selected, including  $SD16INTDLYx$  and  $SD16GAINx$  settings.

The typical temperature sensor transfer function is shown in Figure 21–11. When switching inputs of an SD16 channel to the temperature sensor, adequate delay must be provided using  $SD16INTDLYx$  to allow the digital filter to settle and assure that conversion results are valid. The temperature sensor offset error can be large, and may need to be calibrated for most applications. See device-specific data sheet for temperature sensor parameters.

Figure 21–11. Typical Temperature Sensor Transfer Function



### 21.2.11 Interrupt Handling

The SD16 has 2 interrupt sources for each ADC channel:

- SD16IFG
- SD16OVIFG

The SD16IFG bits are set when their corresponding SD16MEMx memory register is written with a conversion result. An interrupt request is generated if the corresponding SD16IE bit and the GIE bit are set. The SD16 overflow condition occurs when a conversion result is written to any SD16MEMx location before the previous conversion result was read.

### SD16IV, Interrupt Vector Generator

All SD16 interrupt sources are prioritized and combined to source a single interrupt vector. SD16IV is used to determine which enabled SD16 interrupt source requested an interrupt. The highest priority SD16 interrupt request that is enabled generates a number in the SD16IV register (see register description). This number can be evaluated or added to the program counter to automatically enter the appropriate software routine. Disabled SD16 interrupts do not affect the SD16IV value.

Any access, read or write, of the SD16IV register has no effect on the SD16OVIFG or SD16IFG flags. The SD16IFG flags are reset by reading the associated SD16MEMx register or by clearing the flags in software. SD16OVIFG bits can only be reset with software.

If another interrupt is pending after servicing of an interrupt, another interrupt is generated. For example, if the SD16OVIFG and one or more SD16IFG interrupts are pending when the interrupt service routine accesses the SD16IV register, the SD16OVIFG interrupt condition is serviced first and the corresponding flag(s) must be cleared in software. After the RETI instruction of the interrupt service routine is executed, the highest priority SD16IFG pending generates another interrupt request.

### Interrupt Delay Operation

The SD16INTDLYx bits control the timing for the first interrupt service request for the corresponding channel. This feature delays the interrupt request for a completed conversion by up to four conversion cycles allowing the digital filter to settle prior to generating an interrupt request. The delay is applied each time the SD16SC bit is set or when the SD16GAINx or SD16INCHx bits for the channel are modified. SD16INTDLYx disables overflow interrupt generation for the channel for the selected number of delay cycles. Interrupt requests for the delayed conversions are not generated during the delay.

## SD16 Interrupt Handling Software Example

The following software example shows the recommended use of SD16IV and the handling overhead. The SD16IV value is added to the PC to automatically jump to the appropriate routine.

The numbers at the right margin show the necessary CPU cycles for each instruction. The software overhead for different interrupt sources includes interrupt latency and return-from-interrupt cycles, but not the task handling itself. The latencies are:

- SD16OVIFG, CH0 SD16IFG, CH1 SD16IFG                      16 cycles
- CH2 SD16IFG    14 cycles

The interrupt handler for channel 2 SD16IFG shows a way to check immediately if a higher prioritized interrupt occurred during the processing of the ISR. This saves nine cycles if another SD16 interrupt is pending.

```

; Interrupt handler for SD16.
INT_SD16          ; Enter Interrupt Service Routine          6
  ADD    &SD16IV,PC; Add offset to PC                      3
  RETI                   ; Vector 0: No interrupt          5
  JMP    ADOV           ; Vector 2: ADC overflow          2
  JMP    ADM0           ; Vector 4: CH_0 SD16IFG          2
  JMP    ADM1           ; Vector 6: CH_1 SD16IFG          2
;
; Handler for CH_2 SD16IFG starts here. No JMP required.
;
ADM2      MOV    &SD16MEM2,xxx ; Move result, flag is reset
          ...           ; Other instruction needed?
          JMP    INT_SD16     ; Check other int pending    2
;
; Remaining Handlers
;
ADM1      MOV    &SD16MEM1,xxx ; Move result, flag is reset
          ...           ; Other instruction needed?
          RETI                   ; Return                    5
;
ADM0      MOV    &SD16MEM0,xxx ; Move result, flag is reset
          RETI                   ; Return                    5
;
ADOV      ...           ; Handle SD16MEMx overflow
          RETI                   ; Return                    5

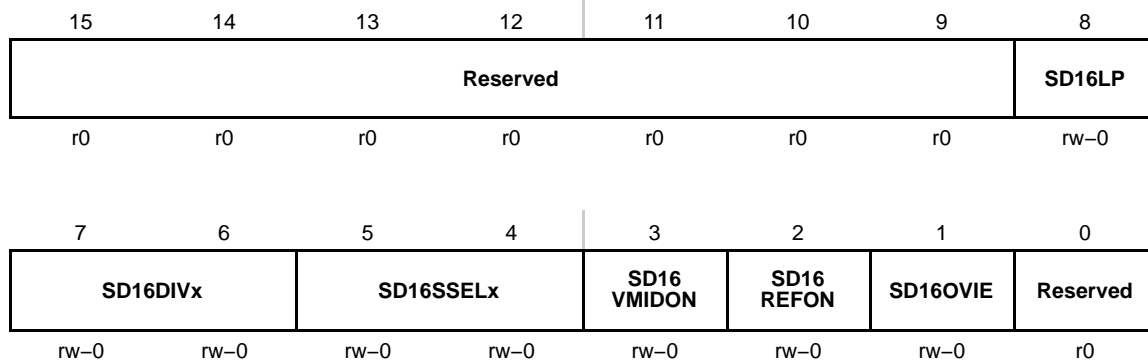
```

## 21.3 SD16 Registers

The SD16 registers are listed in Table 21–3:

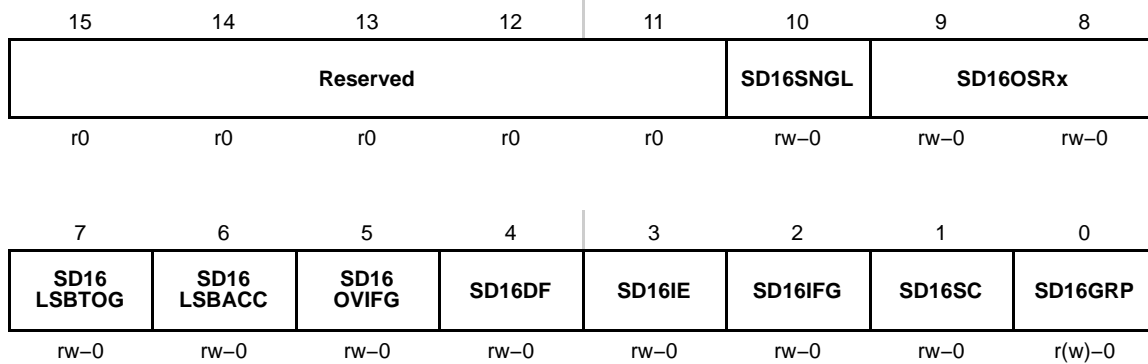
*Table 21–3. SD16 Registers*

<b>Register</b>	<b>Short Form</b>	<b>Register Type</b>	<b>Address</b>	<b>Initial State</b>
SD16 Control	SD16CTL	Read/write	0100h	Reset with PUC
SD16 Interrupt Vector	SD16IV	Read/write	0110h	Reset with PUC
SD16 Channel 0 Control	SD16CCTL0	Read/write	0102h	Reset with PUC
SD16 Channel 0 Conversion Memory	SD16MEM0	Read/write	0112h	Reset with PUC
SD16 Channel 0 Input Control	SD16INCTL0	Read/write	0B0h	Reset with PUC
SD16 Channel 0 Preload	SD16PRE0	Read/write	0B8h	Reset with PUC
SD16 Channel 1 Control	SD16CCTL1	Read/write	0104h	Reset with PUC
SD16 Channel 1 Conversion Memory	SD16MEM1	Read/write	0114h	Reset with PUC
SD16 Channel 1 Input Control	SD16INCTL1	Read/write	0B1h	Reset with PUC
SD16 Channel 1 Preload	SD16PRE1	Read/write	0B9h	Reset with PUC
SD16 Channel 2 Control	SD16CCTL2	Read/write	0106h	Reset with PUC
SD16 Channel 2 Conversion Memory	SD16MEM2	Read/write	0116h	Reset with PUC
SD16 Channel 2 Input Control	SD16INCTL2	Read/write	0B2h	Reset with PUC
SD16 Channel 2 Preload	SD16PRE2	Read/write	0BAh	Reset with PUC

**SD16CTL, SD16 Control Register**

<b>Reserved</b>	Bits 15-9	Reserved
<b>SD16LP</b>	Bit 8	Low power mode. This bit selects a reduced speed, reduced power mode for the SD16. 0 Low-power mode is disabled 1 Low-power mode is enabled. The maximum clock frequency for the SD16 is reduced.
<b>SD16DIVx</b>	Bits 7-6	SD16 clock divider 00 /1 01 /2 10 /4 11 /8
<b>SD16SSELx</b>	Bits 5-4	SD16 clock source select 00 MCLK 01 SMCLK 10 ACLK 11 External TACLK
<b>SD16 VMIDON</b>	Bit 3	$V_{MID}$ buffer on 0 Off 1 On
<b>SD16 REFON</b>	Bit 2	Reference generator on 0 Reference off 1 Reference on
<b>SD16OVIE</b>	Bit 1	SD16 overflow interrupt enable. The GIE bit must also be set to enable the interrupt. 0 Overflow interrupt disabled 1 Overflow interrupt enabled
<b>Reserved</b>	Bit 0	Reserved

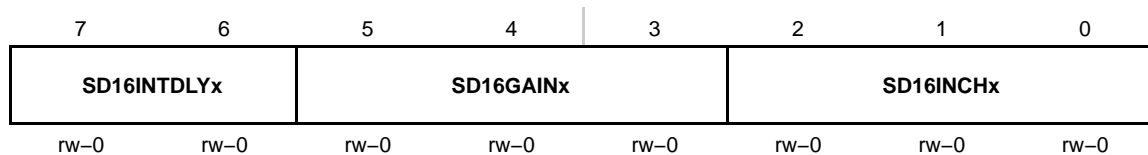
## SD16CTLx, SD16 Channel x Control Register



<b>Reserved</b>	Bits 15-11	Reserved
<b>SD16SNGL</b>	Bit 10	Single conversion mode select 0 Continuous conversion mode 1 Single conversion mode
<b>SD16OSRx</b>	Bits 9-8	Oversampling ratio 00 256 01 128 10 64 11 32
<b>SD16 LSBTOG</b>	Bit 7	LSB toggle. This bit, when set, causes SD16LSBACC to toggle each time the SD16MEMx register is read. 0 SD16LSBACC does not toggle with each SD16MEMx read 1 SD16LSBACC toggles with each SD16MEMx read
<b>SD16 LSBACC</b>	Bit 6	LSB access. This bit allows access to the upper or lower 16-bits of the SD16 conversion result. 0 SD16MEMx contains the most significant 16-bits of the conversion. 1 SD16MEMx contains the least significant 16-bits of the conversion.
<b>SD16OVIFG</b>	Bit 5	SD16 overflow interrupt flag 0 No overflow interrupt pending 1 Overflow interrupt pending
<b>SD16DF</b>	Bit 4	SD16 data format 0 Offset binary 1 2's complement
<b>SD16IE</b>	Bit 3	SD16 interrupt enable 0 Disabled 1 Enabled

<b>SD16IFG</b>	Bit 2	SD16 interrupt flag. SD16IFG is set when new conversion results are available. SD16IFG is automatically reset when the corresponding SD16MEMx register is read, or may be cleared with software. 0 No interrupt pending 1 Interrupt pending
<b>SD16SC</b>	Bit 1	SD16 start conversion 0 No conversion start 1 Start conversion
<b>SD16GRP</b>	Bit 0	SD16 group. Groups SD16 channel with next higher channel. Not used for the last channel. 0 Not grouped 1 Grouped

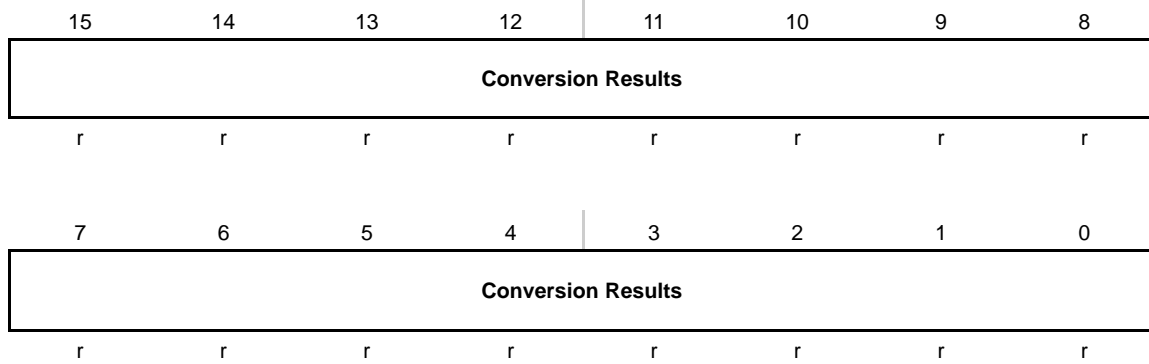
### SD16INCTLx, SD16 Channel x Input Control Register



<b>SD16INTDLYx</b>	Bits 7-6	Interrupt delay generation after conversion start. These bits select the delay for the first interrupt after conversion start. 00 Fourth sample causes interrupt 01 Third sample causes interrupt 10 Second sample causes interrupt 11 First sample causes interrupt
<b>SD16GAINx</b>	Bits 5-3	SD16 preamplifier gain 000 x1 001 x2 010 x4 011 x8 100 x16 101 x32 110 Reserved 111 Reserved
<b>SD16INCHx</b>	Bits 2-0	SD16 channel differential pair input 000 Ax.0 001 Ax.1 010 Ax.2 011 Ax.3 100 Ax.4 101 Ax.5 110 Ax.6- Temperature Sensor 111 Ax.7- Short for PGA offset measurement

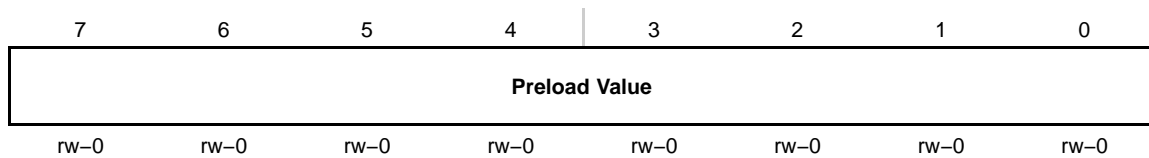


**SD16MEMx, SD16 Channel x Conversion Memory Register**



**Conversion Result**    Bits 15-0    Conversion Results. The SD16MEMx register holds the upper or lower 16-bits of the digital filter output, depending on the SD16LSBACC bit.

**SD16PREx, SD16 Channel x Preload Register**



**SD16 Preload Value**    Bits 7-0    SD16 digital filter preload value.

**SD16IV, SD16 Interrupt Vector Register**

15	14	13	12	11	10	9	8
<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
<b>0</b>	<b>0</b>	<b>0</b>	<b>SD16IVx</b>				<b>0</b>
r0	r0	r0	r-0	r-0	r-0	r-0	r0

**SD16IVx**      Bits      SD16 interrupt vector value  
                   15-0

SD16IV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
000h	No interrupt pending	–	
002h	SD16MEMx overflow	SD16CCTLx SD16OVIFG <sup>†</sup>	Highest
004h	SD16_0 Interrupt	SD16CCTL0 SD16IFG	
006h	SD16_1 Interrupt	SD16CCTL1 SD16IFG	
008h	SD16_2 Interrupt	SD16CCTL1 SD16IFG	
00Ah	Reserved	–	
00Ch	Reserved	–	
00Eh	Reserved	–	
010h	Reserved	–	Lowest

<sup>†</sup> When an SD16 overflow occurs, the user must check all SD16CCTLx SD16OVIFG flags in order to determine which channel overflowed.

**SD16\_A**

---

---

---

---

The SD16\_A module is a single-converter 16-bit, sigma-delta analog-to-digital conversion module with high impedance input buffer. This chapter describes the SD16\_A. The SD16\_A module is implemented in the MSP430F42x0 devices.

<b>Topic</b>	<b>Page</b>
22.1 SD16_A Introduction .....	22-2
22.2 SD16_A Operation .....	22-4
22.3 SD16_A Registers .....	22-14

## 22.1 SD16\_A Introduction

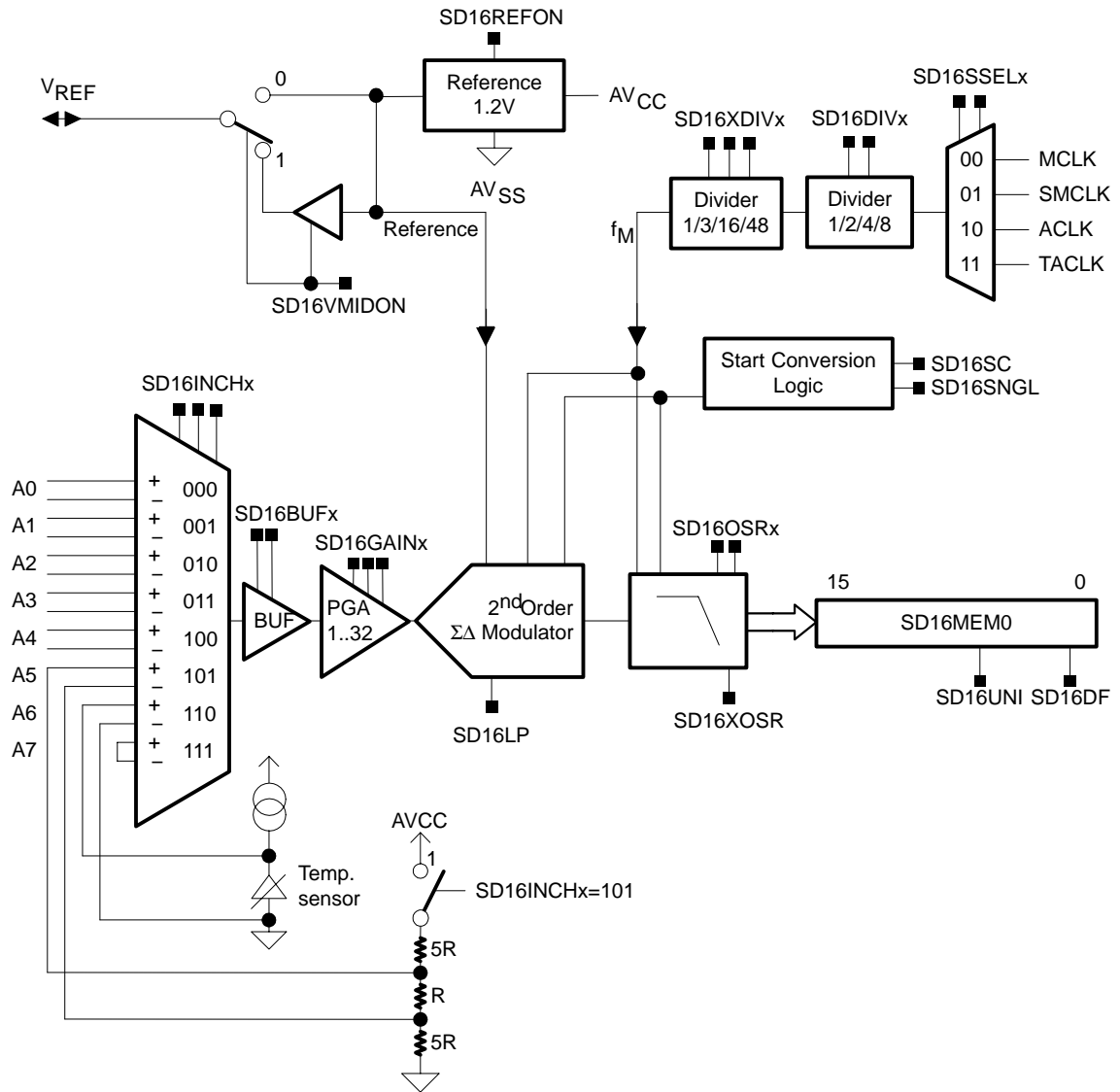
The SD16\_A module consists of one sigma-delta analog-to-digital converter with an high impedance input buffer and an internal voltage reference. It has up to 8 fully differential multiplexed inputs including a built-in temperature sensor. The converter is based on a second-order oversampling sigma-delta modulator and digital decimation filter. The decimation filter is a comb type filter with selectable oversampling ratios of up to 1024. Additional filtering can be done in software.

Features of the SD16\_A include:

- 16-bit sigma-delta architecture
- Up to 8 multiplexed differential analog inputs per channel
- Software selectable on-chip reference voltage generation (1.2V)
- Software selectable internal or external reference
- Built-in temperature sensor
- Up to 1.1 MHz modulator input frequency
- High impedance input buffer
- Selectable low-power conversion mode

The block diagram of the SD16\_A module is shown in Figure 22–1.

Figure 22-1. SD16\_A Block Diagram



## 22.2 SD16\_A Operation

The SD16\_A module is configured with user software. The setup and operation of the SD16\_A is discussed in the following sections.

### 22.2.1 ADC Core

The analog-to-digital conversion is performed by a 1-bit, second-order sigma-delta modulator. A single-bit comparator within the modulator quantizes the input signal with the modulator frequency  $f_M$ . The resulting 1-bit data stream is averaged by the digital filter for the conversion result.

### 22.2.2 Analog Input Range and PGA

The full-scale input voltage range for each analog input pair is dependent on the gain setting of the programmable gain amplifier of each channel. The maximum full-scale range is  $\pm V_{FSR}$  where  $V_{FSR}$  is defined by:

$$V_{FSR} = \frac{V_{REF}/2}{GAIN_{PGA}}$$

For a 1.2V reference, the maximum full-scale input range for a gain of 1 is:

$$\pm V_{FSR} = \frac{1.2V/2}{1} = \pm 0.6V$$

Refer to the device-specific data sheet for full-scale input specifications.

### 22.2.3 Voltage Reference Generator

The SD16\_A module has a built-in 1.2V reference. It is enabled by the SD16REFON bit. When using the internal reference an external 100nF capacitor connected from  $V_{REF}$  to  $AV_{SS}$  is recommended to reduce noise. The internal reference voltage can be used off-chip when SD16VMIDON = 1. The buffered output can provide up to 1mA of drive. When using the internal reference off-chip, a 470nF capacitor connected from  $V_{REF}$  to  $AV_{SS}$  is required. See device-specific data sheet for parameters.

An external voltage reference can be applied to the  $V_{REF}$  input when SD16REFON and SD16VMIDON are both reset.

### 22.2.4 Auto Power-Down

The SD16\_A is designed for low power applications. When the SD16\_A is not actively converting, it is automatically disabled and automatically re-enabled when a conversion is started. The reference is not automatically disabled, but can be disabled by setting SD16REFON = 0. When the SD16\_A or reference are disabled, they consume no current.

### 22.2.5 Channel Selection

The SD16\_A can convert up to 8 differential pair inputs multiplexed into the PGA. Up to five input pairs (A0-A4) are available externally on the device. A resistive divider to measure the supply voltage is available using the A5 multiplexer input. An internal temperature sensor is available using the A6 multiplexer input. Input A7 is a shorted connection between the + and - input pair and can be used to calibrate the offset of the SD16\_A input stage.

### Analog Input Setup

The analog input is configured using the SD16INCTL0 and the SD16AE registers. The SD16INCHx bits select one of eight differential input pairs of the analog multiplexer. The gain for the PGA is selected by the SD16GAINx bits. A total of six gain settings are available. The SD16AEx bits enable or disable the analog input pin. Setting any SD16AEx bit disables the multiplexed digital circuitry for the associated pin. See the device-specific datasheet for pin diagrams.

During conversion any modification to the SD16INCHx and SD16GAINx bits will become effective with the next decimation step of the digital filter. After these bits are modified, the next three conversions may be invalid due to the settling time of the digital filter. This can be handled automatically with the SD16INTDLYx bits. When SD16INTDLY = 00h, conversion interrupt requests will not begin until the 4<sup>th</sup> conversion after a start condition.

The high impedance input buffer can be enabled using the SD16BUFx bits. The speed settings are selected based on the SD16\_A modulator frequency as shown in Table 22–1.

Table 22–1. High Input Impedance Buffer

SD16BUFx	Buffer	SD16 Modulator Frequency $f_M$
00	Buffer disabled	
01	Low speed/current	$f_M < 200\text{kHz}$
10	Medium speed/current	$200\text{kHz} < f_M < 700\text{kHz}$
11	High speed/current	$700\text{kHz} < f_M < 1.1\text{MHz}$

### 22.2.6 Analog Input Characteristics

The SD16 uses a switched-capacitor input stage that appears as an impedance to external circuitry. The equivalent impedance differs for the PGA settings and is given in the device-specific datasheet.

An external RC anti-aliasing filter is recommended for the SD16\_A to prevent aliasing of the input signal. The cutoff frequency should be  $< 10\text{ kHz}$  for a 1 Mhz modulator clock and  $\text{OSR} = 256$ . The cutoff frequency may set to a lower frequency for applications that have lower bandwidth requirements.

## 22.2.7 Digital Filter

The digital filter processes the 1-bit data stream from the modulator using a SINC<sup>3</sup> comb filter. The transfer function is described in the z-Domain by:

$$H(z) = \left( \frac{1}{OSR} \times \frac{1 - z^{-OSR}}{1 - z^{-1}} \right)^3$$

and in the frequency domain by:

$$H(f) = \left[ \frac{\text{sinc}\left(OSR\pi\frac{f}{f_M}\right)}{\text{sinc}\left(\pi\frac{f}{f_M}\right)} \right]^3 = \left[ \frac{1}{OSR} \times \frac{\sin\left(OSR \times \pi \times \frac{f}{f_M}\right)}{\sin\left(\pi \times \frac{f}{f_M}\right)} \right]^3$$

where the oversampling rate, OSR, is the ratio of the modulator frequency  $f_M$  to the sample frequency  $f_S$ . Figure 22–2 shows the filter's frequency response for an OSR of 32. The first filter notch is at  $f_S = f_M/OSR$ . The notch's frequency can be adjusted by changing the modulator's frequency,  $f_M$ , using SD16SSELx and SD16DIVx and the oversampling rate using the SD16OSRx and SD16XOSR bits .

The digital filter for each enabled ADC channel completes the decimation of the digital bit-stream and outputs new conversion results to the SD16MEM0 register at the sample frequency  $f_S$ .

Figure 22–2. Comb Filter's Frequency Response with OSR = 32

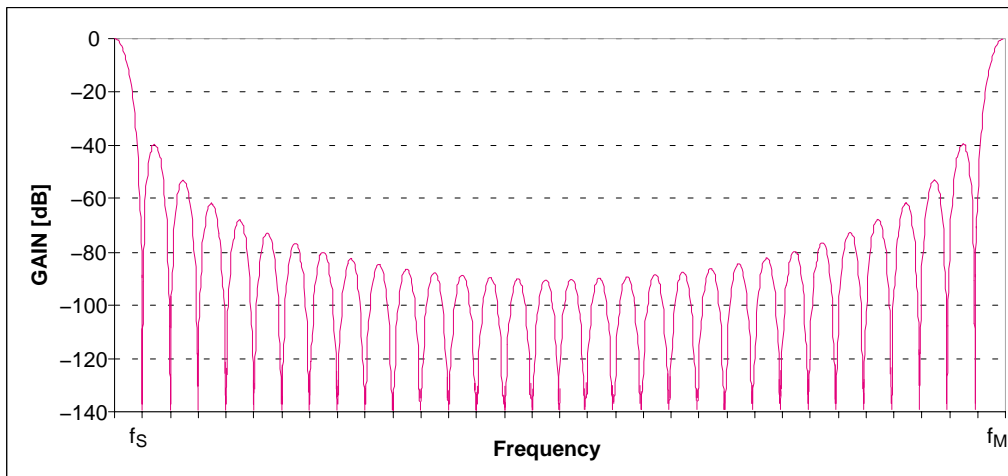
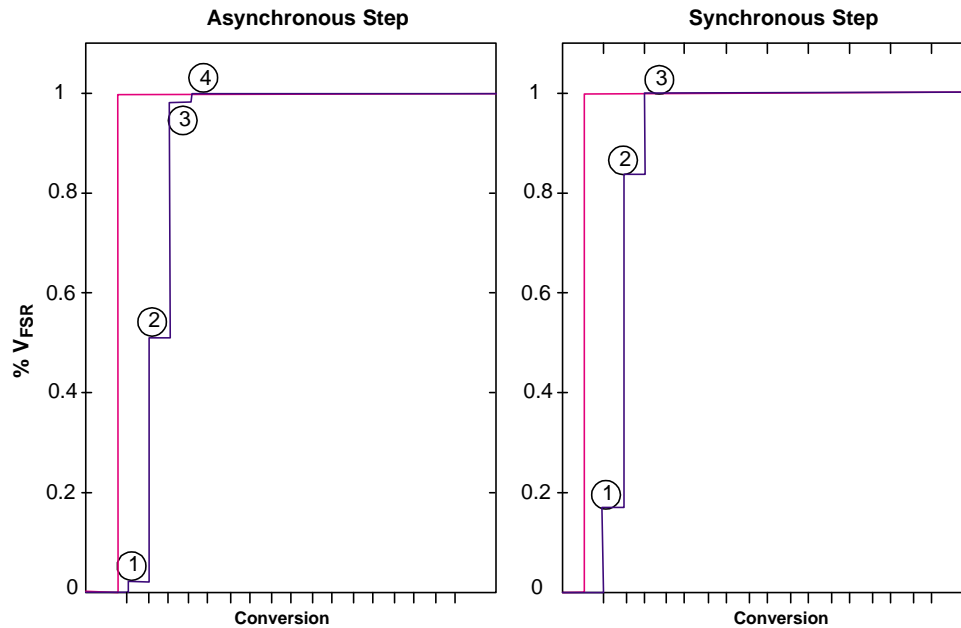




Figure 22–3 shows the digital filter step response and conversion points. For step changes at the input after start of conversion a settling time must be allowed before a valid conversion result is available. The SD16INTDLYx bits can provide sufficient filter settling time for a full-scale change at the ADC input. If the step occurs synchronously to the decimation of the digital filter the valid data will be available on the third conversion. An asynchronous step will require one additional conversion before valid data is available.

Figure 22–3. Digital Filter Step Response and Conversion Points



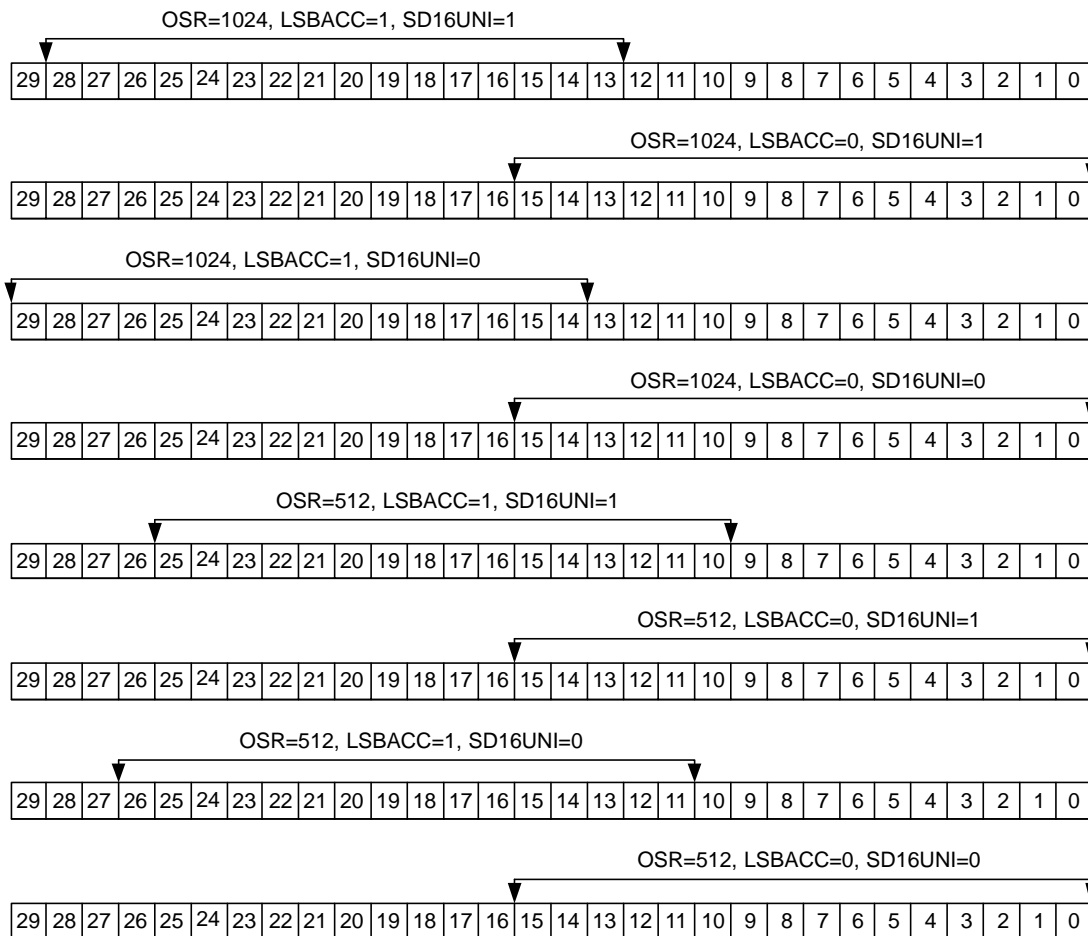
## Digital Filter Output

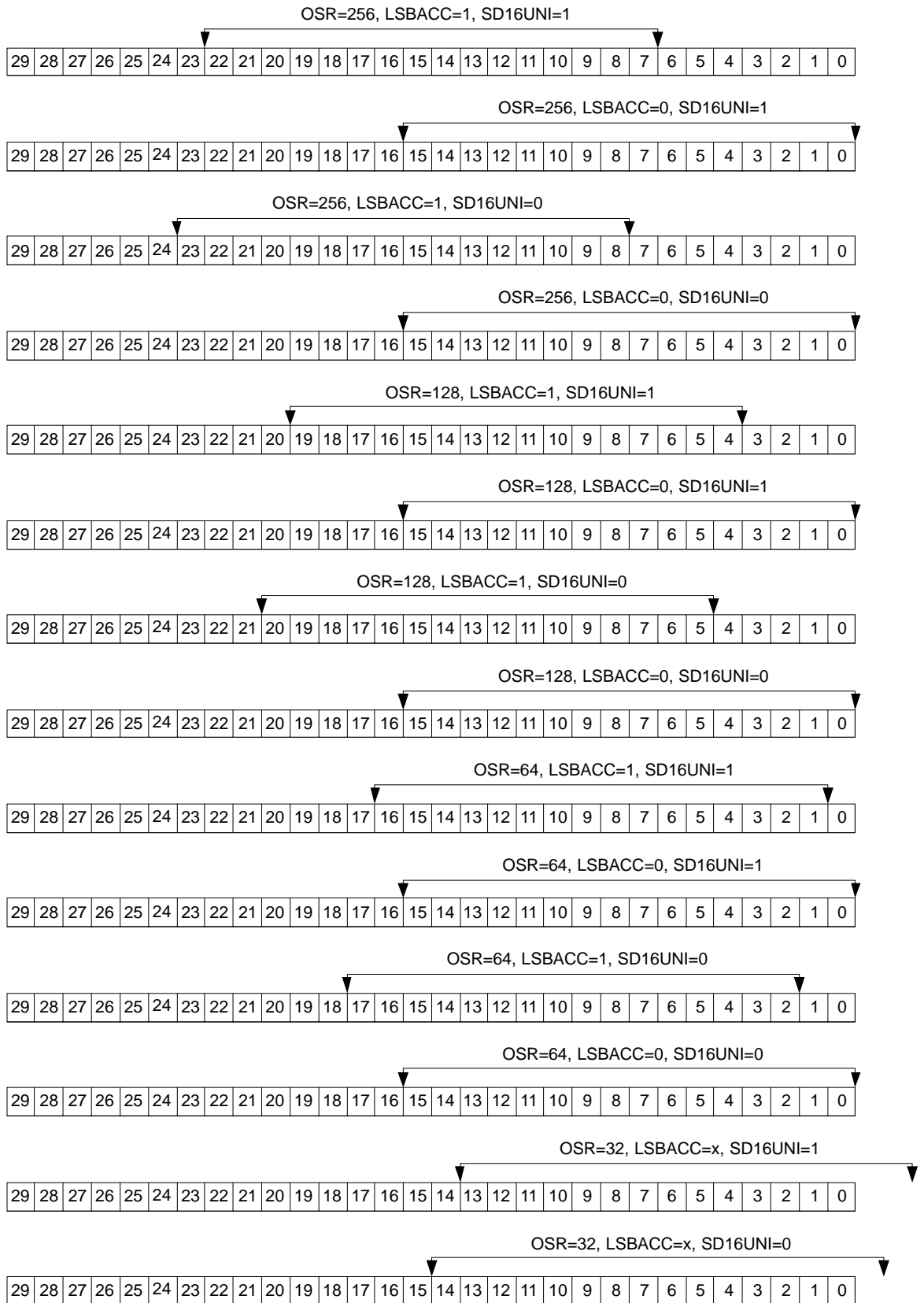
The number of bits output by the digital filter is dependent on the oversampling ratio and ranges from 15 to 30 bits. Figure 22–4 shows the digital filter output and their relation to SD16MEM0 for each OSR, LSBACC, and SD16UNI setting. For example, for OSR = 1024, LSBACC = 1, and SD16UNI = 1, the SD16MEM0 register contains bits 28 – 13 of the digital filter output. When OSR = 32, the one (SD16UNI = 0) or two (SD16UNI=1) LSBs are always zero.

The SD16LSBACC and SD16LSBTOG bits give access to the least significant bits of the digital filter output. When SD16LSBACC = 1 the 16 least significant bits of the digital filter's output are read from SD16MEM0 using word instructions. The SD16MEM0 register can also be accessed with byte instructions returning only the 8 least significant bits of the digital filter output.

When SD16LSBTOG = 1 the SD16LSBACC bit is automatically toggled each time SD16MEM0 is read. This allows the complete digital filter output result to be read with two reads of SD16MEM0. Setting or clearing SD16LSBTOG does not change SD16LSBACC until the next SD16MEM0 access.

Figure 22–4. Used Bits of Digital Filter Output





## 22.2.8 Conversion Memory Register: SD16MEM0

The SD16MEM0 register is associated with the SD16\_A channel. Conversion results are moved to the SD16MEM0 register with each decimation step of the digital filter. The SD16IFG bit is set when new data is written to SD16MEM0. SD16IFG is automatically cleared when SD16MEM0 is read by the CPU or may be cleared with software.

### Output Data Format

The output data format is configurable in two's complement, offset binary or unipolar mode as shown in Table 22–2. The data format is selected by the SD16DF and SD16UNI bits.

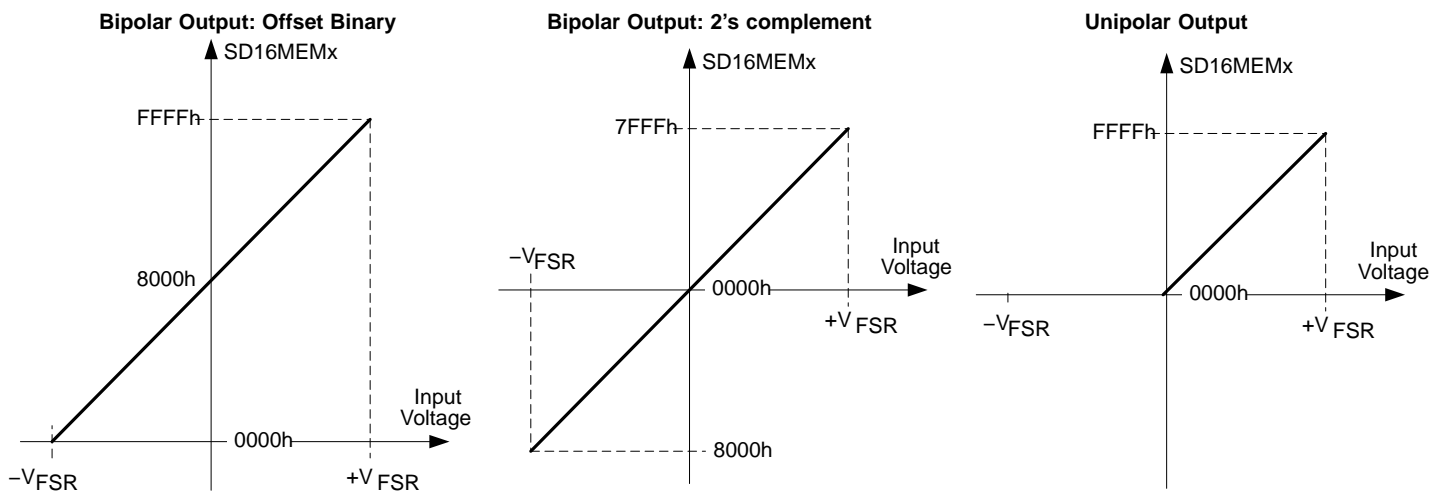
Table 22–2. Data Format

SD16UNI	SD16DF	Format	Analog Input	SD16MEM0 <sup>†</sup>	Digital Filter Output (OSR =256)
0	0	Bipolar Offset Binary	+FSR	FFFF	FFFFFF
			ZERO	8000	800000
			-FSR	0000	000000
0	1	Bipolar Two's compliment	+FSR	7FFF	7FFFFFFF
			ZERO	0000	000000
			-FSR	8000	800000
1	X	Unipolar	+FSR	FFFF	FFFFFF
			ZERO	0000	800000
			-FSR	0000	000000

<sup>†</sup> Independent of SD16OSRx and SD16XOSR settings; SD16LSBACC = 0.

Figure 22–5 shows the relationship between the full-scale input voltage range from  $-V_{FSR}$  to  $+V_{FSR}$  and the conversion result. The data formats are illustrated.

Figure 22–5. Input Voltage vs. Digital Output



## 22.2.9 Conversion Modes

The SD16\_A module can be configured for two modes of operation, listed in Table 22–3. The SD16SNGL bit selects the conversion mode.

Table 22–3. Conversion Mode Summary

SD16SNGL	Mode	Operation
1	Single conversion	The channel is converted once.
0	Continuous conversion	The channel is converted continuously.

### Single Conversion

Setting the SD16SC bit of the channel initiates one conversion on that channel when SD16SNGL = 1. The SD16SC bit will automatically be cleared after conversion completion.

Clearing SD16SC before the conversion is completed immediately stops conversion of the channel, the channel is powered down and the corresponding digital filter is turned off. The value in SD16MEM0 can change when SD16SC is cleared. It is recommended that the conversion data in SD16MEM0 be read prior to clearing SD16SC to avoid reading an invalid result.

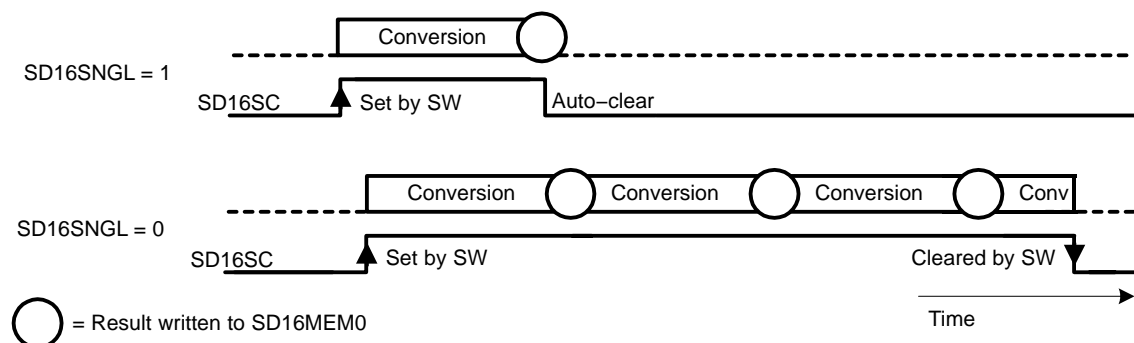
### Continuous Conversion

When SD16SNGL = 0 continuous conversion mode is selected. Conversion of the channel will begin when SD16SC is set and continue until the SD16SC bit is cleared by software.

Clearing SD16SC immediately stops conversion of the selected channel, the channel is powered down and the corresponding digital filter is turned off. The value in SD16MEM0 can change when SD16SC is cleared. It is recommended that the conversion data in SD16MEM0 be read prior to clearing SD16SC to avoid reading an invalid result.

Figure 22–6 shows conversion operation.

Figure 22–6. Single Channel Operation

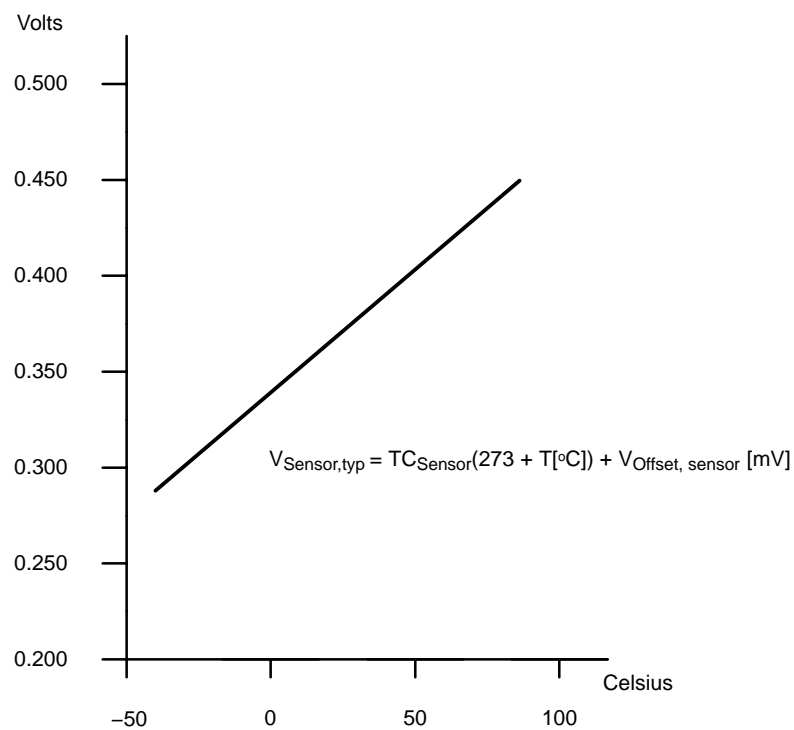


### 22.2.10 Using the Integrated Temperature Sensor

To use the on-chip temperature sensor, the user selects the analog input channel SD16INCHx = 110. Any other configuration is done as if an external channel was selected, including SD16INTDLYx and SD16GAINx settings.

The typical temperature sensor transfer function is shown in Figure 22–7. When switching inputs of an SD16\_A channel to the temperature sensor, adequate delay must be provided using SD16INTDLYx to allow the digital filter to settle and assure that conversion results are valid. The temperature sensor offset error can be large, and may need to be calibrated for most applications. See device-specific data sheet for temperature sensor parameters.

Figure 22–7. Typical Temperature Sensor Transfer Function



### 22.2.11 Interrupt Handling

The SD16\_A has 2 interrupt sources for its ADC channel:

- SD16IFG
- SD16OVIFG

The SD16IFG bit is set when the SD16MEM0 memory register is written with a conversion result. An interrupt request is generated if the corresponding SD16IE bit and the GIE bit are set. The SD16\_A overflow condition occurs when a conversion result is written to SD16MEM0 location before the previous conversion result was read.

### SD16IV, Interrupt Vector Generator

All SD16\_A interrupt sources are prioritized and combined to source a single interrupt vector. SD16IV is used to determine which enabled SD16\_A interrupt source requested an interrupt. The highest priority SD16\_A interrupt request that is enabled generates a number in the SD16IV register (see register description). This number can be evaluated or added to the program counter to automatically enter the appropriate software routine. Disabled SD16\_A interrupts do not affect the SD16IV value.

Any access, read or write, of the SD16IV register has no effect on the SD16OVIFG or SD16IFG flags. The SD16IFG flags are reset by reading the SD16MEM0 register or by clearing the flags in software. SD16OVIFG bits can only be reset with software.

If another interrupt is pending after servicing of an interrupt, another interrupt is generated. For example, if the SD16OVIFG and one or more SD16IFG interrupts are pending when the interrupt service routine accesses the SD16IV register, the SD16OVIFG interrupt condition is serviced first and the corresponding flag(s) must be cleared in software. After the RETI instruction of the interrupt service routine is executed, the highest priority SD16IFG pending generates another interrupt request.

### Interrupt Delay Operation

The SD16INTDLYx bits control the timing for the first interrupt service request for the corresponding channel. This feature delays the interrupt request for a completed conversion by up to four conversion cycles allowing the digital filter to settle prior to generating an interrupt request. The delay is applied each time the SD16SC bit is set or when the SD16GAINx or SD16INCHx bits for the channel are modified. SD16INTDLYx disables overflow interrupt generation for the channel for the selected number of delay cycles. Interrupt requests for the delayed conversions are not generated during the delay.

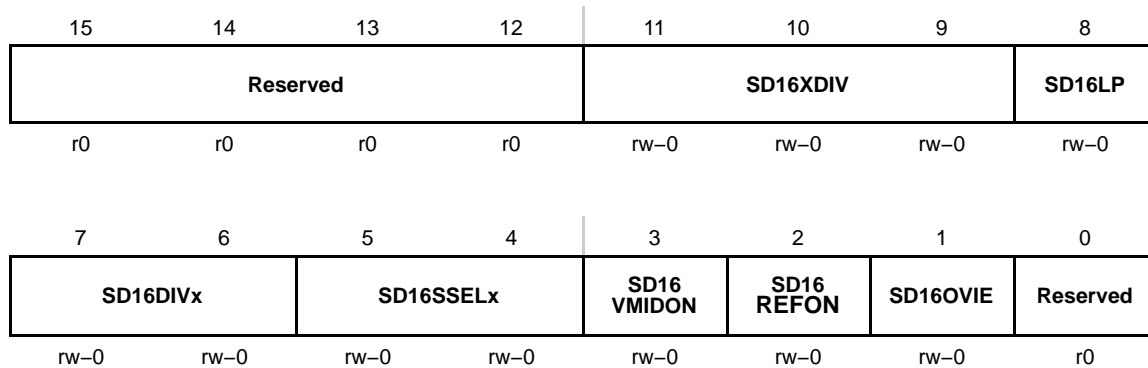
## 22.3 SD16\_A Registers

The SD16\_A registers are listed in Table 22–4:

Table 22–4. SD16\_A Registers

Register	Short Form	Register Type	Address	Initial State
SD16_A Control	SD16CTL	Read/write	0100h	Reset with PUC
SD16_A Interrupt Vector	SD16IV	Read/write	0110h	Reset with PUC
SD16_A Channel 0 Control	SD16CCTL0	Read/write	0102h	Reset with PUC
SD16_A Conversion Memory	SD16MEM0	Read/write	0112h	Reset with PUC
SD16_A Input Control	SD16INCTL0	Read/write	0B0h	Reset with PUC
SD16_A Analog Enable	SD16AE	Read/write	0B7h	Reset with PUC



**SD16CTL, SD16\_A Control Register**

<b>Reserved</b>	Bits 15-12	Reserved
<b>SD16XDIV</b>	Bits 11-9	SD16_A clock divider 000 /1 001 /3 010 /16 011 /48 1xx Reserved
<b>SD16LP</b>	Bit 8	Low power mode. This bit selects a reduced speed, reduced power mode 0 Low-power mode is disabled 1 Low-power mode is enabled. The maximum clock frequency for the SD16_A is reduced.
<b>SD16DIVx</b>	Bits 7-6	SD16_A clock divider 00 /1 01 /2 10 /4 11 /8
<b>SD16SSELx</b>	Bits 5-4	SD16_A clock source select 00 MCLK 01 SMCLK 10 ACLK 11 External TACLK
<b>SD16_A VMIDON</b>	Bit 3	V <sub>MID</sub> buffer on 0 Off 1 On
<b>SD16_A REFON</b>	Bit 2	Reference generator on 0 Reference off 1 Reference on
<b>SD16OVIE</b>	Bit 1	SD16_A overflow interrupt enable. The GIE bit must also be set to enable the interrupt. 0 Overflow interrupt disabled 1 Overflow interrupt enabled
<b>Reserved</b>	Bit 0	Reserved

**SD16CCTL0, SD16\_A Control Register 0**

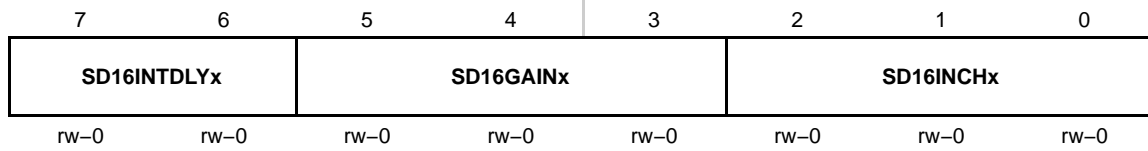
15	14	13	12	11	10	9	8
<b>Reserved</b>	<b>SD16BUFx</b>		<b>SD16UNI</b>	<b>SD16XOSR</b>	<b>SD16SNGL</b>	<b>SD16OSRx</b>	
r0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
<b>SD16 LSBTOG</b>	<b>SD16 LSBACC</b>	<b>SD16 OVIFG</b>	<b>SD16DF</b>	<b>SD16IE</b>	<b>SD16IFG</b>	<b>SD16SC</b>	<b>Reserved</b>
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	r-0

<b>Reserved</b>	Bit 15	Reserved
<b>SD16BUF</b>	Bits 14–13	High impedance input buffer mode 00 Buffer disabled 01 Slow speed/current 10 Medium speed/current 11 High speed/current
<b>SD16UNI</b>	Bit 12	Unipolar mode select 0 Bipolar mode 1 Unipolar mode
<b>SD16XOSR</b>	Bit 11	Extended oversampling ratio. This bit, along with the SD16OSRx bits, select the oversampling ratio. See SD16OSRx bit description for settings.
<b>SD16SNGL</b>	Bit 10	Single conversion mode select 0 Continuous conversion mode 1 Single conversion mode
<b>SD16OSRx</b>	Bits 9–8	Oversampling ratio When SD16XOSR = 0 00 256 01 128 10 64 11 32 When SD16XOSR = 1 00 512 01 1024 10 Reserved 11 Reserved
<b>SD16 LSBTOG</b>	Bit 7	LSB toggle. This bit, when set, causes SD16LSBACC to toggle each time the SD16MEM0 register is read. 0 SD16LSBACC does not toggle with each SD16MEM0 read 1 SD16LSBACC toggles with each SD16MEM0 read

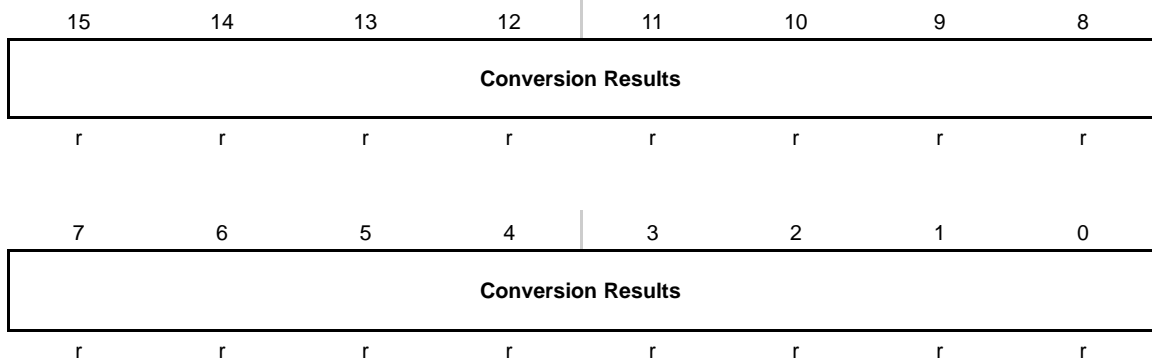
---

<b>SD16 LSBACC</b>	Bit 6	LSB access. This bit allows access to the upper or lower 16-bits of the SD16_A conversion result. 0 SD16MEMx contains the most significant 16-bits of the conversion. 1 SD16MEMx contains the least significant 16-bits of the conversion.
<b>SD16OVIFG</b>	Bit 5	SD16_A overflow interrupt flag 0 No overflow interrupt pending 1 Overflow interrupt pending
<b>SD16DF</b>	Bit 4	SD16_A data format 0 Offset binary 1 2's complement
<b>SD16IE</b>	Bit 3	SD16_A interrupt enable 0 Disabled 1 Enabled
<b>SD16IFG</b>	Bit 2	SD16_A interrupt flag. SD16IFG is set when new conversion results are available. SD16IFG is automatically reset when the corresponding SD16MEMx register is read, or may be cleared with software. 0 No interrupt pending 1 Interrupt pending
<b>SD16SC</b>	Bit 1	SD16_A start conversion 0 No conversion start 1 Start conversion
<b>Reserved</b>	Bit 0	Reserved

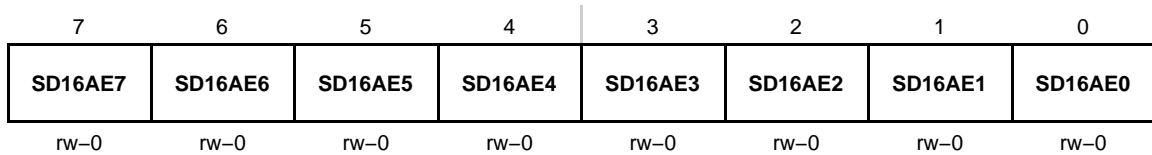
## SD16INCTL0, SD16\_A Input Control Register



<b>SD16INTDLYx</b>	Bits 7-6	<p>Interrupt delay generation after conversion start. These bits select the delay for the first interrupt after conversion start.</p> <p>00 Fourth sample causes interrupt</p> <p>01 Third sample causes interrupt</p> <p>10 Second sample causes interrupt</p> <p>11 First sample causes interrupt</p>
<b>SD16GAINx</b>	Bits 5-3	<p>SD16_A preamplifier gain</p> <p>000 x1</p> <p>001 x2</p> <p>010 x4</p> <p>011 x8</p> <p>100 x16</p> <p>101 x32</p> <p>110 Reserved</p> <p>111 Reserved</p>
<b>SD16INCHx</b>	Bits 2-0	<p>SD16_A channel differential pair input</p> <p>000 A0</p> <p>001 A1</p> <p>010 A2</p> <p>011 A3</p> <p>100 A4</p> <p>101 A5- (<math>AV_{CC} - AV_{SS}</math>) / 11</p> <p>110 A6- Temperature Sensor</p> <p>111 A7- Short for PGA offset measurement</p>

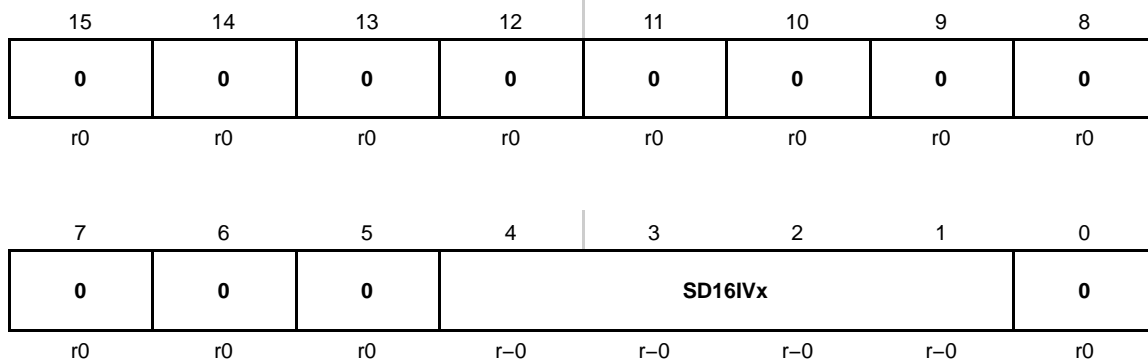
**SD16MEM0, SD16\_A Conversion Memory Register**

**Conversion Result** Bits 15-0 Conversion Results. The SD16MEMx register holds the upper or lower 16-bits of the digital filter output, depending on the SD16LSBACC bit.

**SD16AE, SD16\_A Analog Input Enable Register**

**SD16AEx** Bits 7-0 SD16\_A analog enable  
 0 External input disabled. Negative inputs are internally connected to VSS.  
 1 External input enabled.

## SD16IV, SD16\_A Interrupt Vector Register



**SD16IVx**    Bits    SD16\_A interrupt vector value  
                  15-0

SD16IV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
000h	No interrupt pending	–	
002h	SD16MEMx overflow	SD16CCTLx SD16OVIFG	Highest
004h	SD16_A Interrupt	SD16CCTL0 SD16IFG	
006h	Reserved	–	
008h	Reserved	–	
00Ah	Reserved	–	
00Ch	Reserved	–	
00Eh	Reserved	–	
010h	Reserved	–	Lowest

**DAC12**

---

---

---

---

The DAC12 module is a 12-bit, voltage output digital-to-analog converter. This chapter describes the DAC12. Two DAC12 modules are implemented in the MSP430FG43x devices. Only DAC12\_0 is implemented in MSP430x42x0 devices.

<b>Topic</b>	<b>Page</b>
<b>23.1 DAC12 Introduction</b> .....	<b>23-2</b>
<b>23.2 DAC12 Operation</b> .....	<b>23-5</b>
<b>23.3 DAC12 Registers</b> .....	<b>23-11</b>

## 23.1 DAC12 Introduction

The DAC12 module is a 12-bit, voltage output DAC. The DAC12 can be configured in 8- or 12-bit mode and may be used in conjunction with the DMA controller. When multiple DAC12 modules are present, they may be grouped together for synchronous update operation.

Features of the DAC12 include:

- 12-bit monotonic output
- 8- or 12-bit voltage output resolution
- Programmable settling time vs power consumption
- Internal or external reference selection
- Straight binary or 2's complement data format
- Self-calibration option for offset correction
- Synchronized update capability for multiple DAC12s

---

**Note: Multiple DAC12 Modules**

Some devices may integrate more than one DAC12 module. In the case where more than one DAC12 is present on a device, the multiple DAC12 modules operate identically.

Throughout this chapter, nomenclature appears such as DAC12\_xDAT or DAC12\_xCTL to describe register names. When this occurs, the x is used to indicate which DAC12 module is being discussed. In cases where operation is identical, the register is simply referred to as DAC12\_xCTL.

---

The block diagram of the two DAC12 modules in the MSP430FG43x devices is shown in Figure 23–1. The block diagram for the DAC12 module in the MSP430x42x0 devices is shown in Figure 23–2.



Figure 23-1. DAC12 Block Diagram

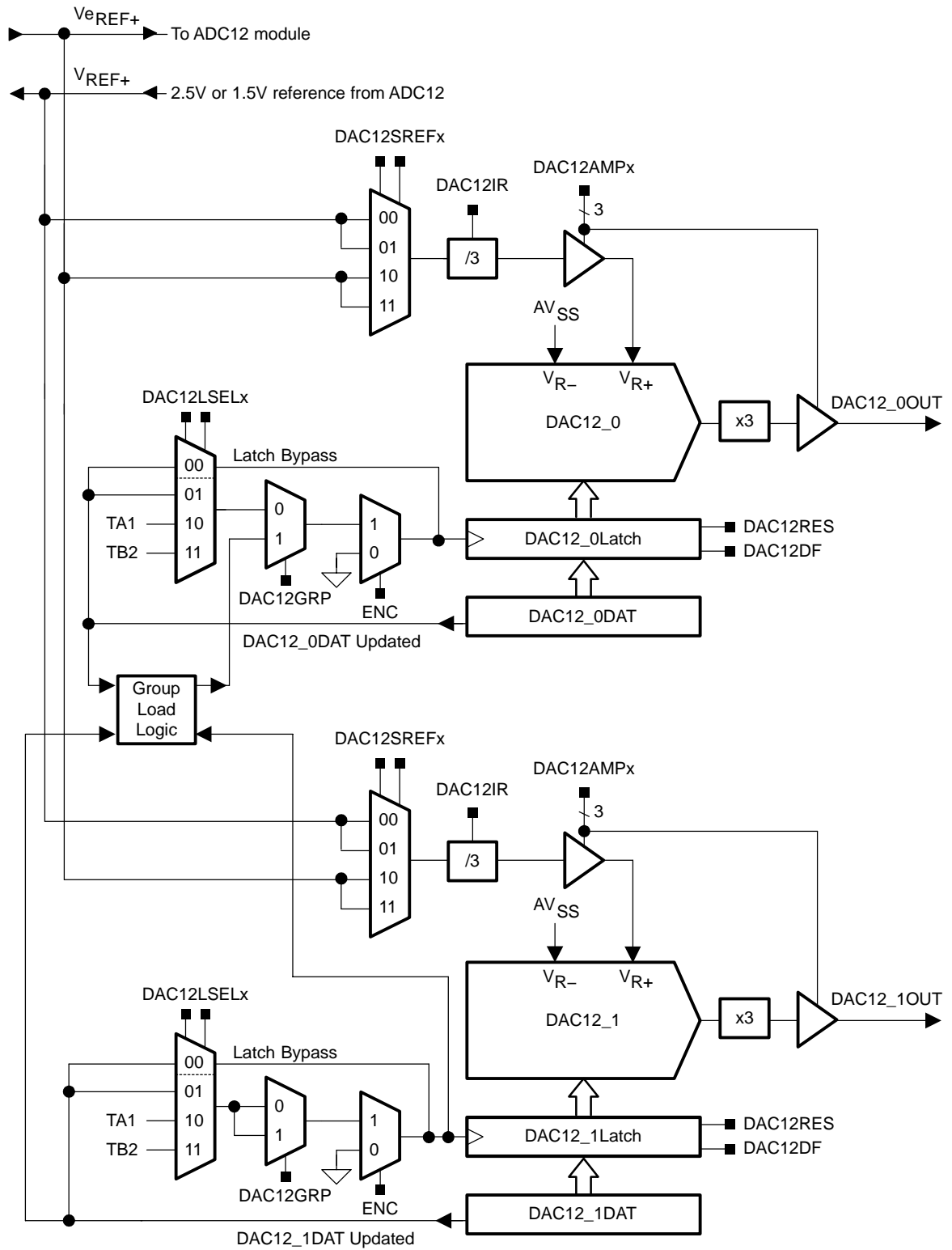
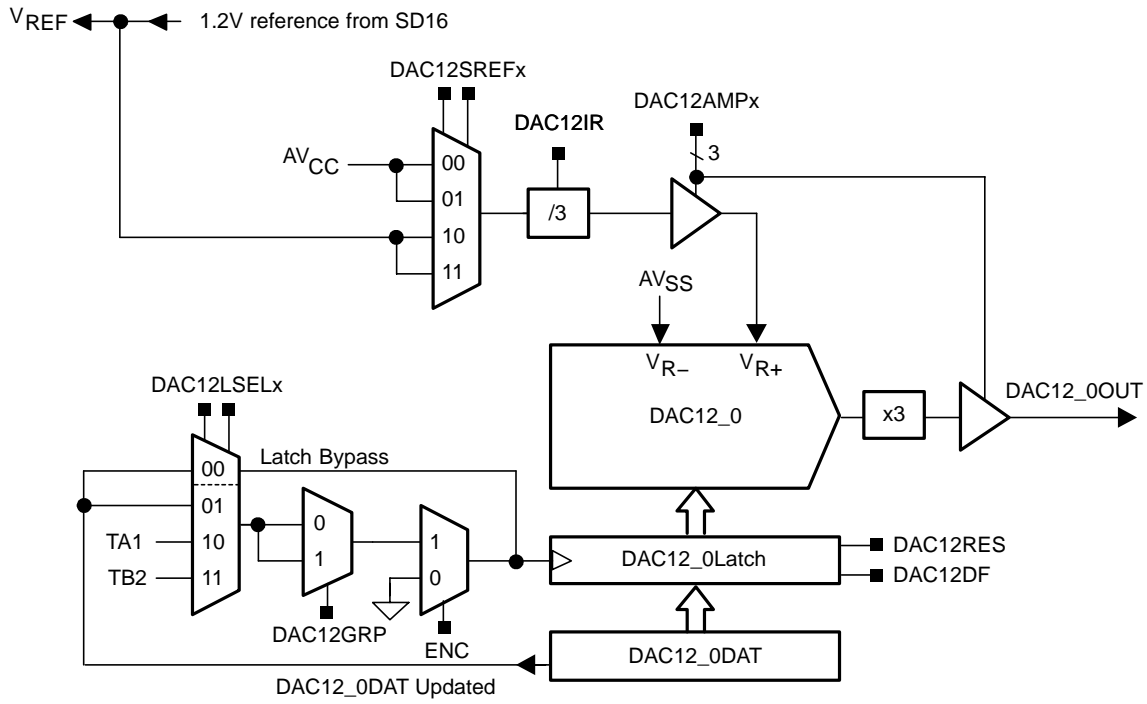


Figure 23–2. DAC12 Block Diagram For MSPx42x0 Devices



## 23.2 DAC12 Operation

The DAC12 module is configured with user software. The setup and operation of the DAC12 is discussed in the following sections.

### 23.2.1 DAC12 Core

The DAC12 can be configured to operate in 8- or 12-bit mode using the DAC12RES bit. The full-scale output is programmable to be 1x or 3x the selected reference voltage via the DAC12IR bit. This feature allows the user to control the dynamic range of the DAC12. The DAC12DF bit allows the user to select between straight binary data and 2's complement data for the DAC. When using straight binary data format, the formula for the output voltage is given in Table 23–1.

Table 23–1. DAC12 Full-Scale Range ( $V_{ref} = V_{eREF+}$  or  $V_{REF+}$ )

Resolution	DAC12RES	DAC12IR	Output Voltage Formula
12 bit	0	0	$V_{out} = V_{ref} \times 3 \times \frac{DAC12\_xDAT}{4096}$
12 bit	0	1	$V_{out} = V_{ref} \times \frac{DAC12\_xDAT}{4096}$
8 bit	1	0	$V_{out} = V_{ref} \times 3 \times \frac{DAC12\_xDAT}{256}$
8 bit	1	1	$V_{out} = V_{ref} \times \frac{DAC12\_xDAT}{256}$

In 8-bit mode the maximum useable value for DAC12\_xDAT is 0FFh and in 12-bit mode the maximum useable value for DAC12\_xDAT is 0FFFh. Values greater than these may be written to the register, but all leading bits are ignored.

### DAC12 Port Selection

On MSP430FG43x devices, the DAC12 outputs are multiplexed with the port P6 pins and ADC12 analog inputs, and also the VeREF+ and P5.1/S0/A12 pins. When DAC12AMPx > 0, the DAC12 function is automatically selected for the pin, regardless of the state of the associated PxSELx and PxDIRx bits. The DAC12OPS bit selects between the P6 pins and the VeREF+ and P5.1 pins for the DAC outputs. For example, when DAC12OPS = 0, DAC12\_0 outputs on P6.6 and DAC12\_1 outputs on P6.7. When DAC12OPS = 1, DAC12\_0 outputs on VeREF+ and DAC12\_1 outputs on P5.1. See the port pin schematic in the device-specific datasheet for more details.

On MSP430x42x0 devices, the DAC12 output is multiplexed with the port P1.4/A3– pin. In this case, the DAC12OPS bit selects the DAC function for the pin. See the port pin schematic in the device-specific datasheet for more details.

### 23.2.2 DAC12 Reference

On MSP430FG43x devices, the reference for the DAC12 is configured to use either an external reference voltage or the internal 1.5-V/2.5-V reference from the ADC12 module with the DAC12SREFx bits. When DAC12SREFx = {0,1} the  $V_{REF+}$  signal is used as the reference and when DAC12SREFx = {2,3} the  $V_{eREF+}$  signal is used as the reference.

On MSP430x42x0 devices, the reference for the DAC12 is configured to use  $AV_{CC}$ , an external reference voltage, or the 1.2-V reference from the SD16 module with the DAC12SREFx bits. When DAC12SREFx = {0,1}  $AV_{CC}$  is used as the reference and when DAC12SREFx = {2,3} the  $V_{REF}$  signal is used as the reference.

To use an ADC internal reference, it must be enabled and configured via the applicable ADC control bits.

### DAC12 Reference Input and Voltage Output Buffers

The reference input and voltage output buffers of the DAC12 can be configured for optimized settling time vs power consumption. Eight combinations are selected using the DAC12AMPx bits. In the low/low setting, the settling time is the slowest, and the current consumption of both buffers is the lowest. The medium and high settings have faster settling times, but the current consumption increases. See the device-specific data sheet for parameters.

### 23.2.3 Updating the DAC12 Voltage Output

The DAC12\_xDAT register can be connected directly to the DAC12 core or double buffered. The trigger for updating the DAC12 voltage output is selected with the DAC12LSELx bits.

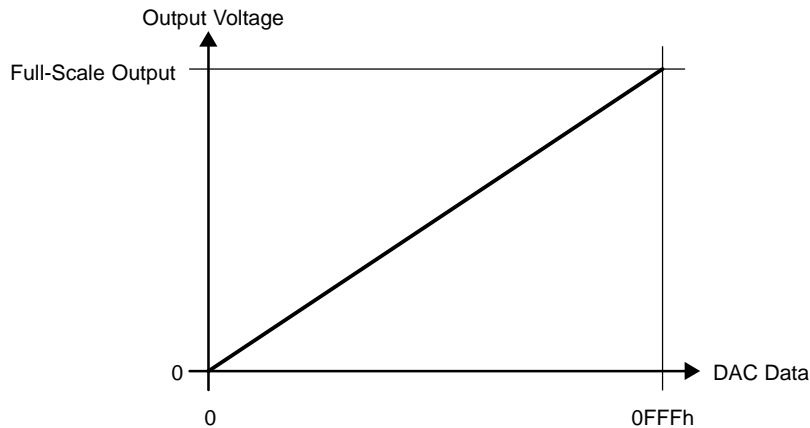
When DAC12LSELx = 0 the data latch is transparent and the DAC12\_xDAT register is applied directly to the DAC12 core. the DAC12 output updates immediately when new DAC12 data is written to the DAC12\_xDAT register, regardless of the state of the DAC12ENC bit.

When DAC12LSELx = 1, DAC12 data is latched and applied to the DAC12 core after new data is written to DAC12\_xDAT. When DAC12LSELx = 2 or 3, data is latched on the rising edge from the Timer\_A CCR1 output or Timer\_B CCR2 output respectively. DAC12ENC must be set to latch the new data when DAC12LSELx > 0.

### 23.2.4 DAC12\_xDAT Data Format

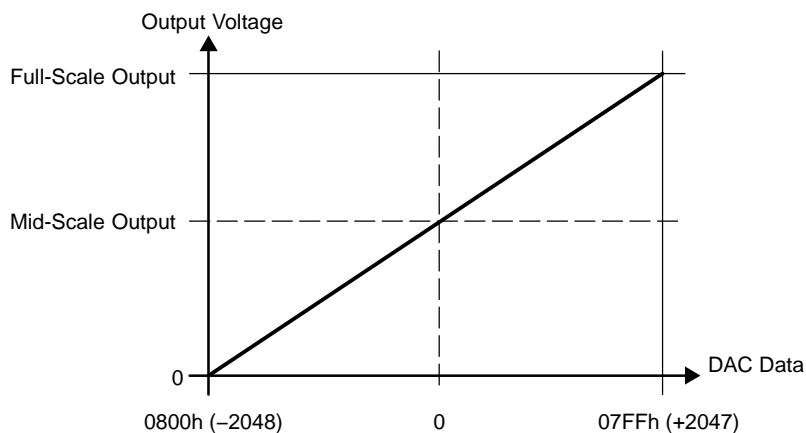
The DAC12 supports both straight binary and 2's complement data formats. When using straight binary data format, the full-scale output value is 0FFFh in 12-bit mode (0FFh in 8-bit mode) as shown in Figure 23–3.

Figure 23–3. Output Voltage vs DAC12 Data, 12-Bit, Straight Binary Mode



When using 2's complement data format, the range is shifted such that a DAC12\_xDAT value of 0800h (0080h in 8-bit mode) results in a zero output voltage, 0000h is the mid-scale output voltage, and 07FFh (007Fh for 8-bit mode) is the full-scale voltage output as shown in Figure 23–4.

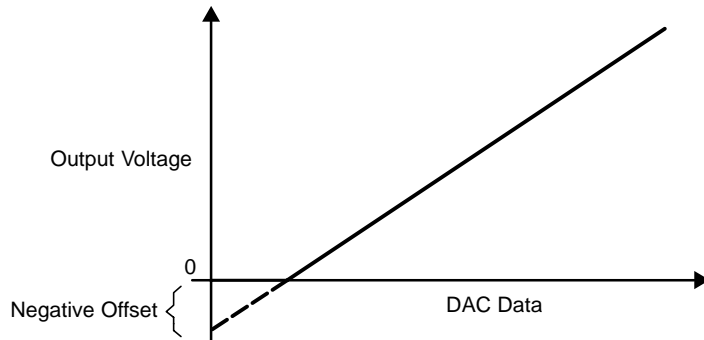
Figure 23–4. Output Voltage vs DAC12 Data, 12-Bit, 2's Complement Mode



### 23.2.5 DAC12 Output Amplifier Offset Calibration

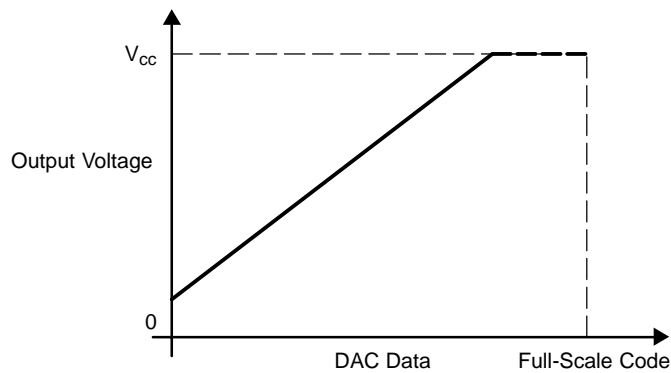
The offset voltage of the DAC12 output amplifier can be positive or negative. When the offset is negative, the output amplifier attempts to drive the voltage negative, but cannot do so. The output voltage remains at zero until the DAC12 digital input produces a sufficient positive output voltage to overcome the negative offset voltage, resulting in the transfer function shown in Figure 23–5.

Figure 23–5. Negative Offset



When the output amplifier has a positive offset, a digital input of zero does not result in a zero output voltage. The DAC12 output voltage reaches the maximum output level before the DAC12 data reaches the maximum code. This is shown in Figure 23–6.

Figure 23–6. Positive Offset



The DAC12 has the capability to calibrate the offset voltage of the output amplifier. Setting the DAC12CALON bit initiates the offset calibration. The calibration should complete before using the DAC12. When the calibration is complete, the DAC12CALON bit is automatically reset. The DAC12AMPx bits should be configured before calibration. For best calibration results, port and CPU activity should be minimized during calibration.

### 23.2.6 Grouping Multiple DAC12 Modules

Multiple DAC12s can be grouped together with the DAC12GRP bit to synchronize the update of each DAC12 output. Hardware ensures that all DAC12 modules in a group update simultaneously independent of any interrupt or NMI event.

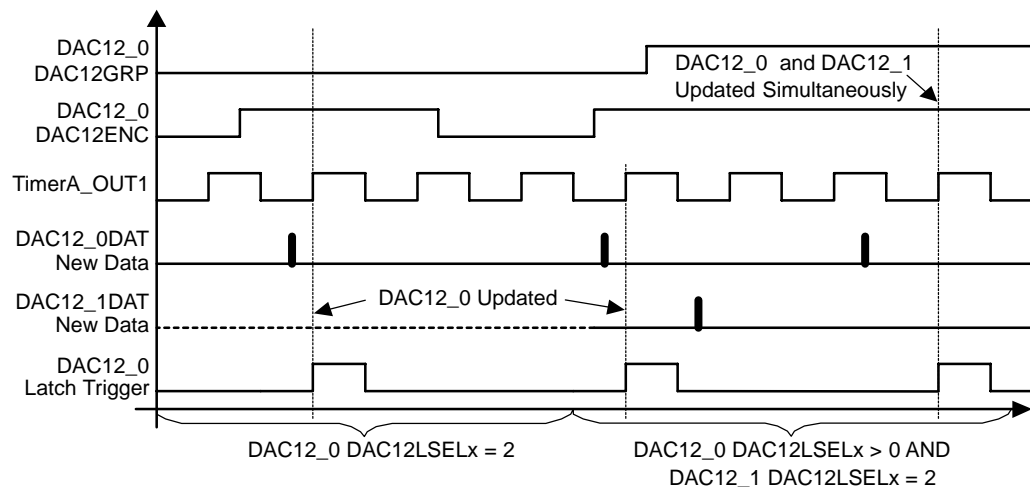
On the MSP430FG43x devices, DAC12\_0 and DAC12\_1 are grouped by setting the DAC12GRP bit of DAC12\_0. The DAC12GRP bit of DAC12\_1 is don't care. When DAC12\_0 and DAC12\_1 are grouped:

- The DAC12\_1 DAC12LSELx bits select the update trigger for both DACs
- The DAC12LSELx bits for both DACs must be > 0
- The DAC12ENC bits of both DACs must be set to 1

When DAC12\_0 and DAC12\_1 are grouped, both DAC12\_xDAT registers must be written to before the outputs update - even if data for one or both of the DACs is not changed. Figure 23–7 shows a latch-update timing example for grouped DAC12\_0 and DAC12\_1.

When DAC12\_0 DAC12GRP = 1 and both DAC12\_x DAC12LSELx > 0 and either DAC12ENC = 0, neither DAC12 will update.

Figure 23–7. DAC12 Group Update Example, Timer\_A3 Trigger



#### Note: DAC12 Settling Time

The DMA controller is capable of transferring data to the DAC12 faster than the DAC12 output can settle. The user must assure the DAC12 settling time is not violated when using the DMA controller. See the device-specific data sheet for parameters.

### 23.2.7 DAC12 Interrupts

The DAC12 interrupt vector is shared with the DMA controller. Software must check the DAC12IFG and DMAIFG flags to determine the source of the interrupt.

The DAC12IFG bit is set when DAC12LSELx > 0 and DAC12 data is latched from the DAC12\_xDAT register into the data latch. When DAC12LSELx = 0, the DAC12IFG flag is not set.

A set DAC12IFG bit indicates that the DAC12 is ready for new data. If both the DAC12IE and GIE bits are set, the DAC12IFG generates an interrupt request. The DAC12IFG flag is not reset automatically. It must be reset by software.



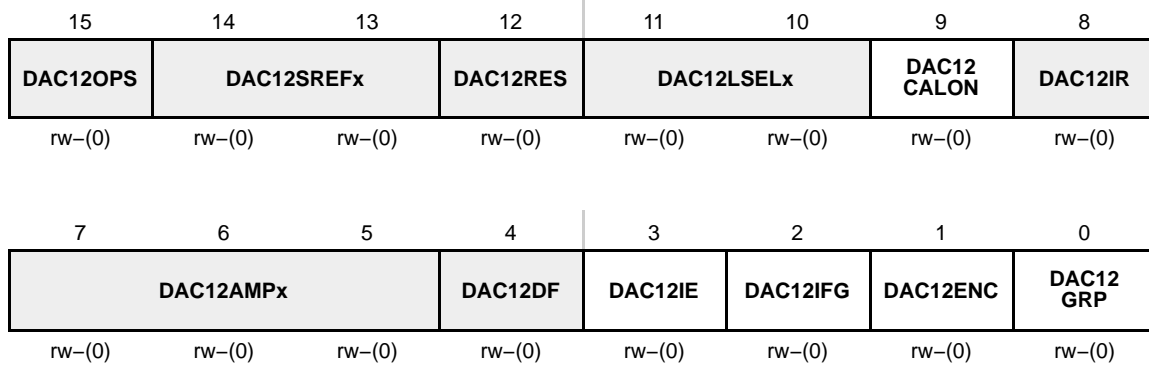
### 23.3 DAC12 Registers

The DAC12 registers are listed in Table 23–2.

*Table 23–2. DAC12 Registers*

<b>Register</b>	<b>Short Form</b>	<b>Register Type</b>	<b>Address</b>	<b>Initial State</b>
DAC12_0 control	DAC12_0CTL	Read/write	01C0h	Reset with POR
DAC12_0 data	DAC12_0DAT	Read/write	01C8h	Reset with POR
DAC12_1 control	DAC12_1CTL	Read/write	01C2h	Reset with POR
DAC12_1 data	DAC12_1DAT	Read/write	01CAh	Reset with POR

**DAC12\_xCTL, DAC12 Control Register**



Modifiable only when DAC12ENC = 0

**DAC12OPS** Bit 15 DAC12 output select for MSP430FG43x and MSP430x42x0 devices. This bit is reserved on all other devices.  
 MSP430FG43x Devices:  
 0 DAC12\_0 output on P6.6, DAC12\_1 output on P6.7  
 1 DAC12\_0 output on V<sub>REF+</sub>, DAC12\_1 output on P5.1  
 MSP430x42x0 Devices:  
 0 DAC12\_0 output not available external to the device  
 1 DAC12\_0 output available internally and externally.

**DAC12 SREFx** Bits 14-13 DAC12 select reference voltage  
 MSP430FG43x Devices:  
 00 V<sub>REF+</sub>  
 01 V<sub>REF+</sub>  
 10 V<sub>REF+</sub>  
 11 V<sub>REF+</sub>  
 MSP430x42x0 Devices:  
 00 AV<sub>CC</sub>  
 01 AV<sub>CC</sub>  
 10 V<sub>REF</sub> (internal from SD16\_A or external)  
 11 V<sub>REF</sub> (internal from SD16\_A or external)

**DAC12 RES** Bit 12 DAC12 resolution select  
 0 12-bit resolution  
 1 8-bit resolution

**DAC12 LSELx** Bits 11-10 DAC12 load select. Selects the load trigger for the DAC12 latch. DAC12ENC must be set for the DAC to update, except when DAC12LSELx = 0.  
 00 DAC12 latch loads when DAC12\_xDAT written (DAC12ENC is ignored)  
 01 DAC12 latch loads when DAC12\_xDAT written, or, when grouped, when all DAC12\_xDAT registers in the group have been written.  
 10 Rising edge of Timer\_A.OUT1 (TA1)  
 11 Rising edge of Timer\_B.OUT2 (TB2)

<b>DAC12 CALON</b>	Bit 9	DAC12 calibration on. This bit initiates the DAC12 offset calibration sequence and is automatically reset when the calibration completes. 0 Calibration is not active 1 Initiate calibration/calibration in progress
<b>DAC12IR</b>	Bit 8	DAC12 input range. This bit sets the reference input and voltage output range. 0 DAC12 full-scale output = 3x reference voltage 1 DAC12 full-scale output = 1x reference voltage
<b>DAC12 AMPx</b>	Bits 7-5	DAC12 amplifier setting. These bits select settling time vs. current consumption for the DAC12 input and output amplifiers.

<b>DAC12AMPx</b>	<b>Input Buffer</b>	<b>Output Buffer</b>
000	Off	DAC12 off, output high Z
001	Off	DAC12 off, output 0 V
010	Low speed/current	Low speed/current
011	Low speed/current	Medium speed/current
100	Low speed/current	High speed/current
101	Medium speed/current	Medium speed/current
110	Medium speed/current	High speed/current
111	High speed/current	High speed/current

<b>DAC12DF</b>	Bit 4	DAC12 data format 0 Straight binary 1 2's compliment
<b>DAC12IE</b>	Bit 3	DAC12 interrupt enable 0 Disabled 1 Enabled
<b>DAC12IFG</b>	Bit 2	DAC12 Interrupt flag 0 No interrupt pending 1 Interrupt pending
<b>DAC12 ENC</b>	Bit 1	DAC12 enable conversion. This bit enables the DAC12 module when DAC12LSELx > 0. when DAC12LSELx = 0, DAC12ENC is ignored. 0 DAC12 disabled 1 DAC12 enabled
<b>DAC12 GRP</b>	Bit 0	DAC12 group. Groups DAC12_x with the next higher DAC12_x. Not used for DAC12_1 on MSP430FG43x devices or on MSP430x42x0 devices. 0 Not grouped 1 Grouped

**DAC12\_xDAT, DAC12 Data Register**



- Unused** Bits 15-12: Unused. These bits are always 0 and do not affect the DAC12 core.
- DAC12 Data** Bits 11-0: DAC12 data

DAC12 Data Format	DAC12 Data
12-bit binary	The DAC12 data are right-justified. Bit 11 is the MSB.
12-bit 2's complement	The DAC12 data are right-justified. Bit 11 is the MSB (sign).
8-bit binary	The DAC12 data are right-justified. Bit 7 is the MSB. Bits 11-8 are don't care and do not effect the DAC12 core.
8-bit 2's complement	The DAC12 data are right-justified. Bit 7 is the MSB (sign). Bits 11-8 are don't care and do not effect the DAC12 core.

# Scan IF

---

---

---

---

The Scan IF peripheral automatically scans sensors and measures linear or rotational motion. This chapter describes the Scan interface. The Scan IF is implemented in the MSP430FW42x devices.

<b>Topic</b>	<b>Page</b>
<b>24.1 Scan IF Introduction</b> .....	<b>24-2</b>
<b>24.2 Scan IF Operation</b> .....	<b>24-4</b>
<b>24.3 Scan IF Registers</b> .....	<b>24-35</b>

## 24.1 Scan IF Introduction

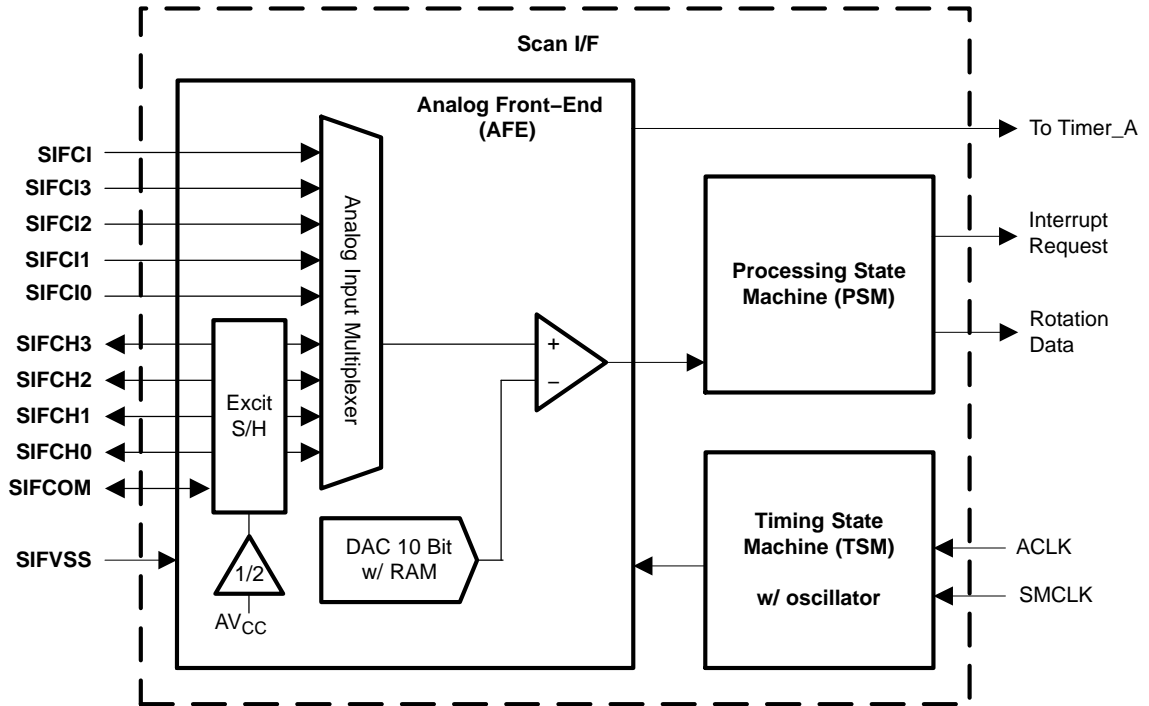
The Scan IF module is used to automatically measure linear or rotational motion with the lowest possible power consumption. The Scan IF consists of three blocks: the analog front end (AFE), the processing state machine (PSM), and the timing state machine (TSM). The analog front end stimulates the sensors, senses the signal levels and converts them into their digital representation. The digital signals are passed into the processing state machine. The processing state machine is used to analyze and count rotation or motion. The timing state machine controls the analog front end and the processing state machine.

The Scan IF features include:

- Support for different types of LC sensors
- Measurement of sensor signal envelope
- Measurement of sensor signal oscillation amplitude
- Support for resistive sensors such as Hall-effect or giant magneto-resistive (GMR) sensors
- Direct analog input for A/D conversion
- Direct digital input for digital sensors such as optical decoders
- Support for quadrature decoding

The Scan IF module block diagram is shown in Figure 24–1.

Figure 24-1. Scan IF Block Diagram



## 24.2 Scan IF Operation

The Scan IF is configured with user software. The setup and operation of the Scan IF is discussed in the following sections.

### 24.2.1 Scan IF Analog Front End

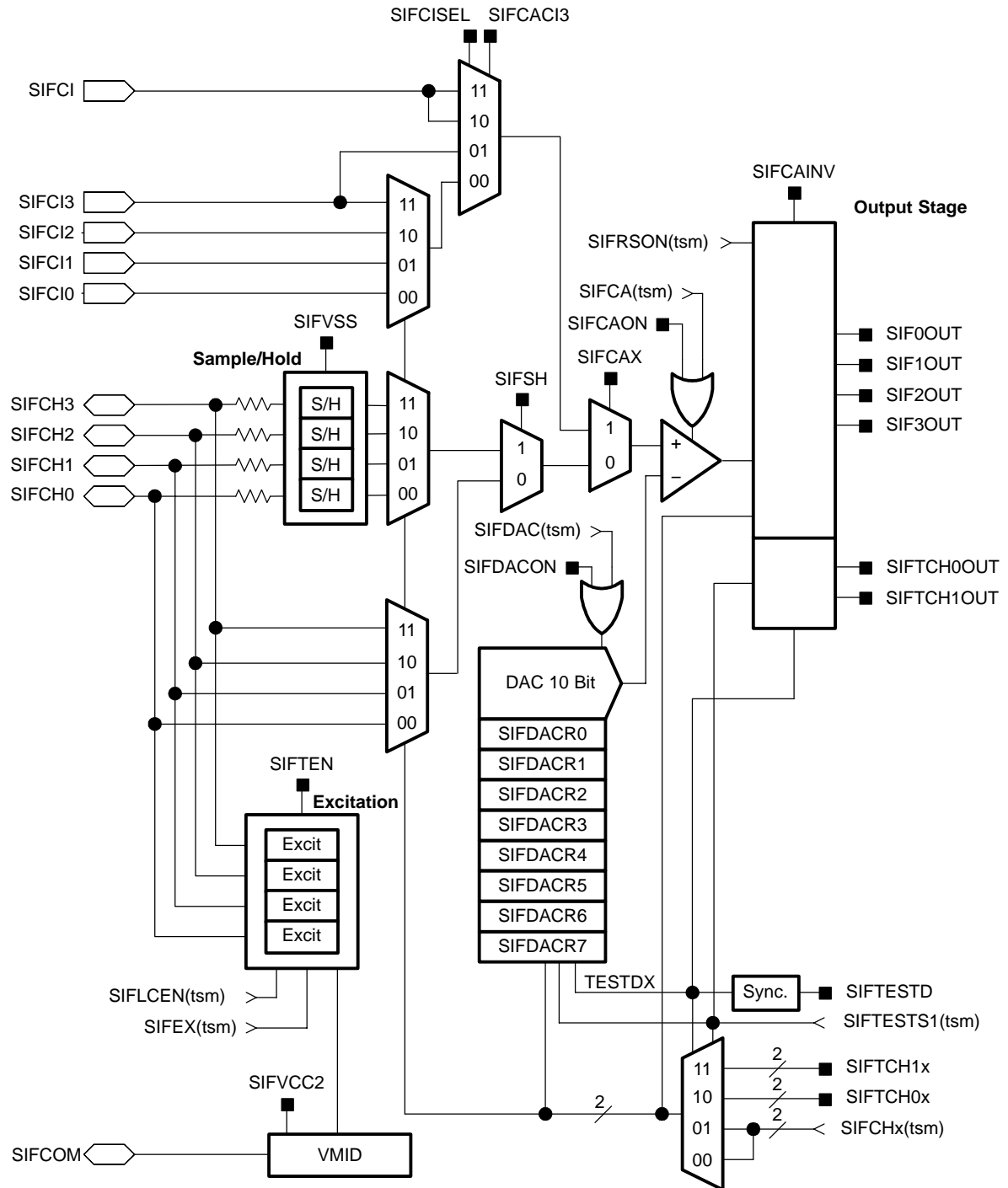
The Scan IF analog front end provides sensor excitation and measurement. The analog front end is automatically controlled by the timing state machine according to the information in the timing state machine table. The analog front end block diagram is shown in Figure 24–2.

**Note: Timing State Machine Signals**

Throughout this chapter, signals from the TSM are noted in the signal name with (tsm). For example, The signal SIFEX(tsm) comes from the TSM.



Figure 24-2. Scan IF Analog Front End Block Diagram



## Excitation

The excitation circuitry is used to excite the LC sensors or to power the resistor dividers. The excitation circuitry is shown in Figure 24–3 for one LC sensor connected. When the SIFTEN bit is set and the SIFSH bit is cleared the excitation circuitry is enabled and the sample-and-hold circuitry is disabled.

When the SIFEX(tsm) signal from the timing state machine is high the SIFCHx input of the selected channel is connected to SIFV<sub>SS</sub> and the SIFCOM input is connected to the mid-voltage generator to excite the sensor. The SIFLCEN(tsm) signal must be high for excitation. While one channel is excited and measured all other channels are automatically disabled. Only the selected channel is excited and measured.

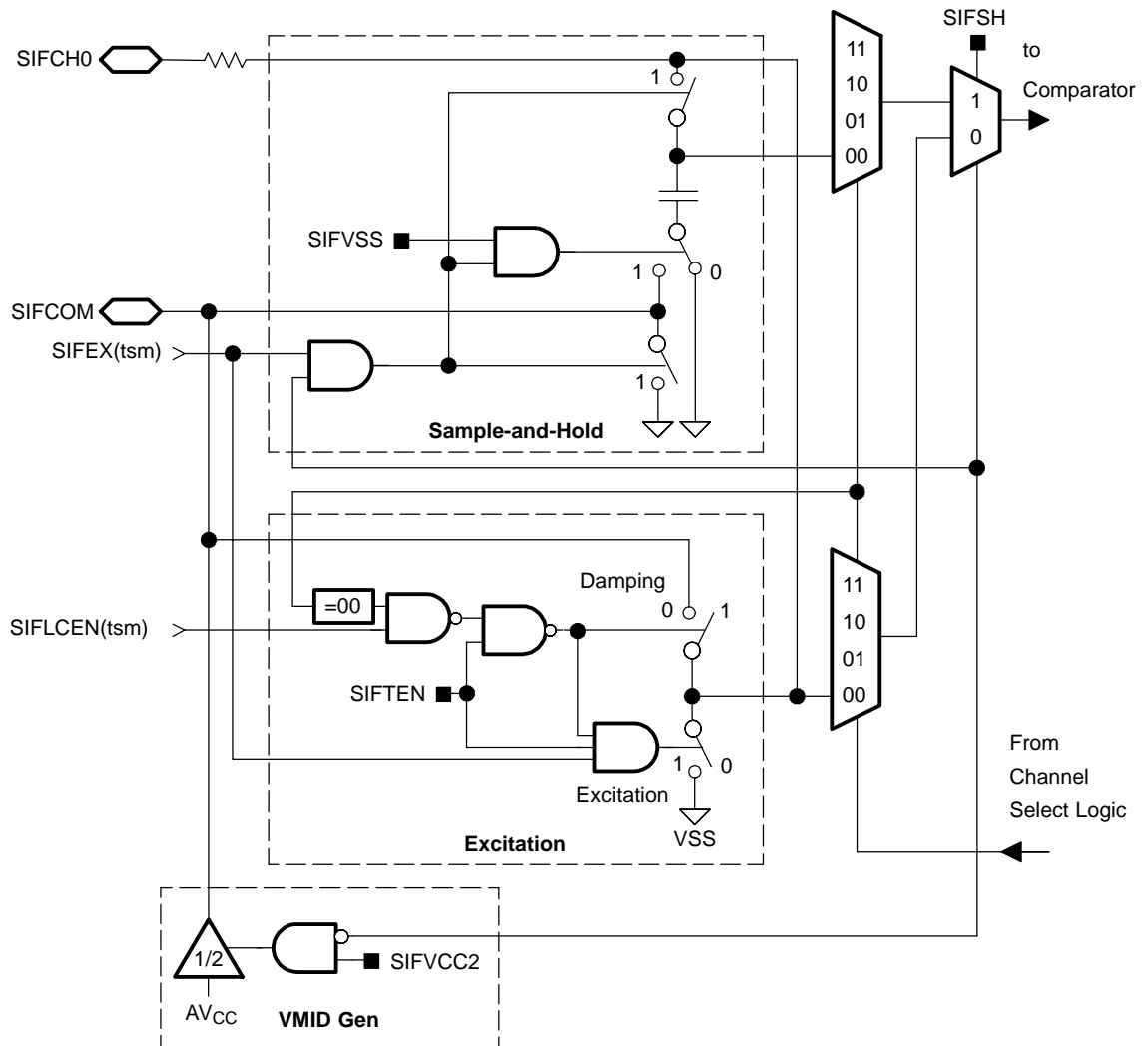
The excitation period should be long enough to overload the LC sensor slightly. After excitation the SIFCHx input is released from ground when SIFEX(tsm) = 0 and the LC sensor can oscillate freely. The oscillations will swing above the positive supply but will be clipped by the protection diode to the positive supply voltage plus one diode drop. This gives consistent maximum oscillation amplitude.

At the end of the measurement the sensor should be damped by setting SIFLCEN(tsm) = 0 to remove any residual energy before the next measurement.

## Mid-Voltage Generator

The mid-voltage generator is on when SIFVCC2 = 1 and allows the LC sensors to oscillate freely. The mid-voltage generator requires a maximum of 6 ms to settle and requires ACLK to be active and operating at 32768 Hz.

Figure 24-3. Excitation and Sample-And-Hold Circuitry



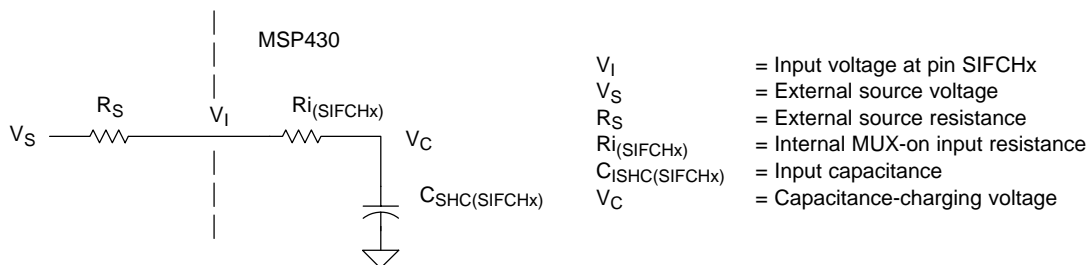
## Sample-And-Hold

The sample-and-hold is used to sample the sensor voltage to be measured. The sample-and-hold circuitry is shown in Figure 24–3. When SIFSH = 1 and SIFTEN = 0 the sample-and-hold circuitry is enabled and the excitation circuitry and mid-voltage generator are disabled. The sample-and-hold is used for resistive dividers or for other analog signals that should be sampled.

Up to four resistor dividers can be connected to SIFCHx and SIFCOM.  $AV_{CC}$  and SIFCOM are the common positive and negative potentials for all connected resistor dividers. When SIFEX(tsm) = 1, SIFCOM is connected to SIFV<sub>SS</sub> allowing current to flow through the dividers. This charges the capacitors of each sample-and-hold circuit to the divider voltages. All resistor divider channels are sampled simultaneously. When SIFEX(tsm) = 0 the sample-and-hold capacitor is disconnected from the resistor divider and SIFCOM is disconnected from SIFV<sub>SS</sub>. After sampling, each channel can be measured sequentially using the channel select logic, the comparator, and the DAC.

The selected SIFCHx input can be modeled as an RC low-pass filter during the sampling time  $t_{\text{sample}}$ , as shown below in Figure 24–4. An internal MUX-on input resistance  $R_{i(\text{SIFCHx})}$  (max. 3 k $\Omega$ ) in series with capacitor  $C_{\text{SCH}(\text{SIFCHx})}$  (max. 7 pF) is seen by the resistor-divider. The capacitor voltage  $V_C$  must be charged to within  $\frac{1}{2}$  LSB of the resistor divider voltage for an accurate 10-bit conversion. See the device-specific datasheet for parameters.

Figure 24–4. Analog Input Equivalent Circuit



The resistance of the source  $R_S$  and  $R_{i(\text{SIFCHx})}$  affect  $t_{\text{sample}}$ . The following equation can be used to calculate the minimum sampling time  $t_{\text{sample}}$  for a 10-bit conversion:

$$t_{\text{sample}} > (R_S + R_{i(\text{SIFCHx})}) \times \ln(2^{11}) \times C_{\text{SHC}(\text{SIFCHx})} \quad (1)$$

Substituting the values for  $R_i$  and  $C_i$  given above, the equation becomes:

$$t_{\text{sample}} > (R_S + 3\text{k}) \times 7.625 \times 7\text{pF} \quad (2)$$

For example, if  $R_S$  is 10 k $\Omega$ ,  $t_{\text{sample}}$  must be greater than 684 ns.

## Direct Analog And Digital Inputs

By setting the SIFCAX bit, external analog or digital signals can be connected directly to the comparator through the SIFClx inputs. This allows measurement capabilities for optical encoders and other sensors.

## Comparator Input Selection And Output Bit Selection

The SIFCAX and SIFSH bits select between the SIFClx channels and the SIFCHx channels for the comparator input as described in Table 24–1.

Table 24–1. SIFCAX and SIFSH Input Selection

SIFCAX	SIFSH	Operation
0	0	SIFCHx and excitation circuitry is selected
0	1	SIFCHx and sample-and-hold circuitry is selected
1	X	SIFClx inputs are selected

The TESTDX signal and SIFTESTS1(tsm) signal select between the SIFxOUT output bits and the SIFTCHxOUT output bits for the comparator output as described in Table 24–2. TESTDX is controlled by the SIFTESTD bit.

Table 24–2. Selected Output Bits

TESTDX	SIFCH(tsm)	SIFTESTS1(tsm)	Selected Output Bit
0	00	X	SIF0OUT
0	01	X	SIF1OUT
0	10	X	SIF2OUT
0	11	X	SIF3OUT
1	X	0	SIFTCH0OUT
1	X	1	SIFTCH1OUT

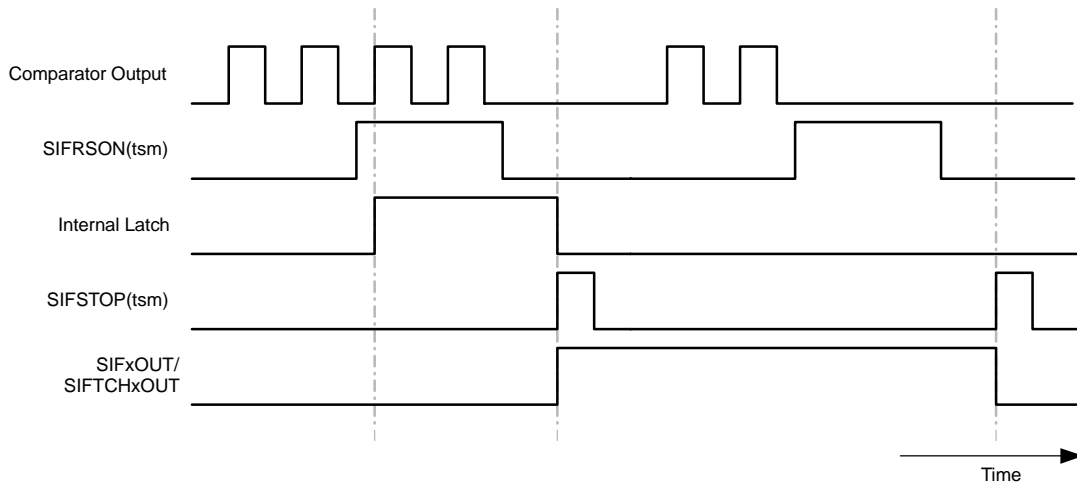
When TESTDX = 0, the SIFCHx(tsm) signals select which SIFClx or SIFCHx channel is excited and connected to the comparator. The SIFCHx(tsm) signals also select the corresponding output bit for the comparator result.

When TESTDX = 1, channel selection depends on the SIFTESTS1(tsm) signal. When TESTDX = 1 and SIFTESTS1(tsm) = 0, input channel selection is controlled with the SIFTCH0x bits and the output bit is SIFTCH0OUT. When TESTDX = 1 and SIFTESTS1(tsm) = 1, input channel selection is controlled with the SIFTCH1x bits and the output bit is SIFTCH1OUT.

When SIFCAX = 1, the SIFCSEL and SIFCI3 bits select between the SIFCIx channels and the SIFCI input allowing storage of the comparator output for one input signal into the four output bits SIF0OUT - SIF3OUT. This can be used to observe the envelope function of sensors.

The output logic is enabled by the SIFRSON(tsm) signal. When the comparator output is high while SIFRSON = 1, an internal latch is set. Otherwise the latch is reset. The latch output is written into the selected output bit with the rising edge of the SIFSTOP(tsm) signal as shown in Figure 24–5.

Figure 24–5. Analog Front-End Output Timing



**Comparator and DAC**

The analog input signals are converted into digital signals by the comparator and the programmable 10-bit DAC. The comparator compares the selected analog signal to a reference voltage generated by the DAC. If the voltage is above the reference the comparator output will be high. Otherwise it will be low. The comparator output can be inverted by setting SIFCAINV. The comparator output is stored in the selected output bit and processed by the processing state machine to detect motion and direction.

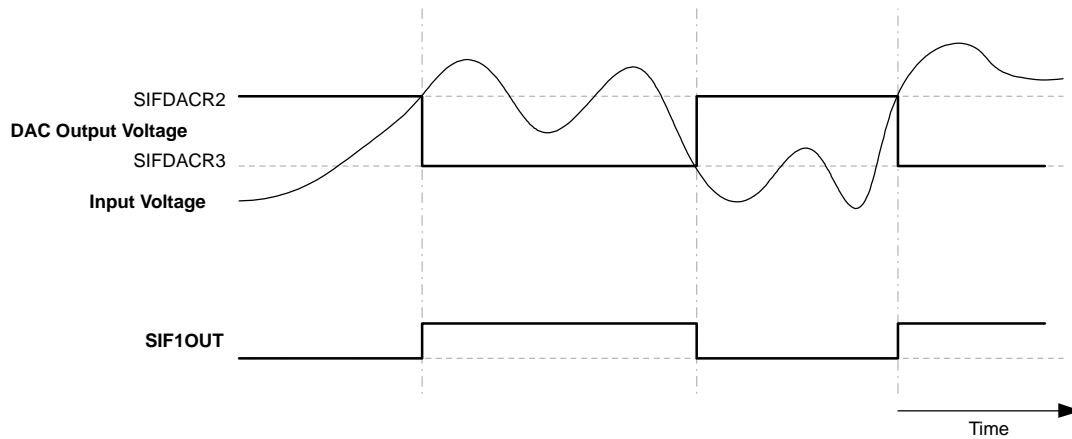
The comparator and the DAC are turned on and off by SIFCA(tsm) signal and the SIFDAC(tsm) signal when needed by the timing state machine. They can also be permanently enabled by setting the SIFCAON and SIFDACON bits. During sensitive measurements enabling the comparator and DAC with the SIFCAON and SIFDACON bits may improve resolution.

For each input there are two DAC registers to set the reference level as listed in Table 24–3. Together with the last stored output of the comparator, SIFxOUT, the two levels can be used as an analog hysteresis as shown in Figure 24–6. The individual settings for the four inputs can be used to compensate for mismatches between the sensors.

Table 24–3. Selected DAC Registers

Selected Output Bit, SIFxOUT	Last Value of SIFxOUT	DAC Register Used
SIF0OUT	0	SIFDACR0
	1	SIFDACR1
SIF1OUT	0	SIFDACR2
	1	SIFDACR3
SIF2OUT	0	SIFDACR4
	1	SIFDACR5
SIF3OUT	0	SIFDACR6
	1	SIFDACR7

Figure 24–6. Analog Hysteresis With DAC Registers



When TESTDX = 1, the SIFDACR6 and SIFDACR7 registers are used as the comparator reference as described in Table 24–4.

Table 24–4. DAC Register Select When TESTDX = 1

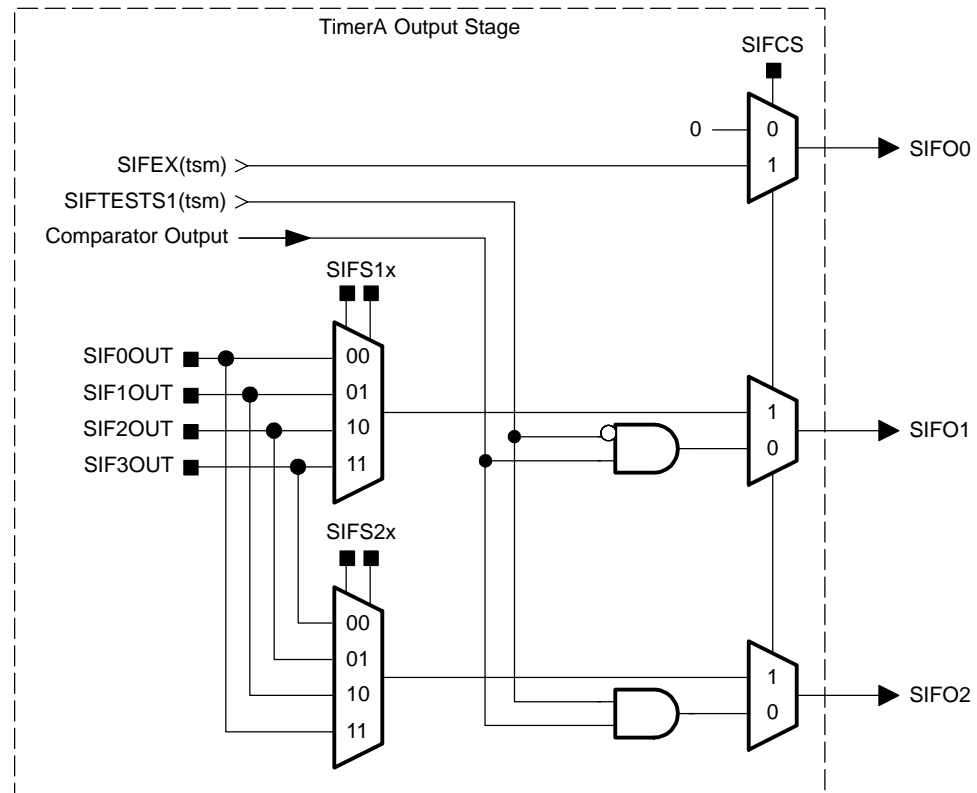
SIFTESTS1(tsm)	DAC Register Used
0	SIFDACR6
1	SIFDACR7



## Internal Signal Connections to Timer1\_A5

The outputs of the analog front end are connected to 3 different capture/compare registers of Timer1\_A5. The output stage of the analog front end, shown in Figure 24–7. provides two different modes that are selected by the SIFCS bit and provides the SIF0x signals to Timer1\_A5. See the device-specific datasheet for connection of these signals.

Figure 24–7. TimerA Output Stage of the Analog Front End



When SIFCS = 1, the SIFEX(tsm) signal and the comparator output can be selected as inputs to different Timer1\_A5 capture/compare registers. This can be used to measure the time between excitation of a sensor and the last oscillation that passes through the comparator or to perform a slope A/D conversion.

When SIFCS = 0, the output bits SIFxOUT can be selected as inputs to Timer1\_A5 with the SIFS1x and SIFS2x bits. This can be used to measure the duty cycle of SIFxOUT.

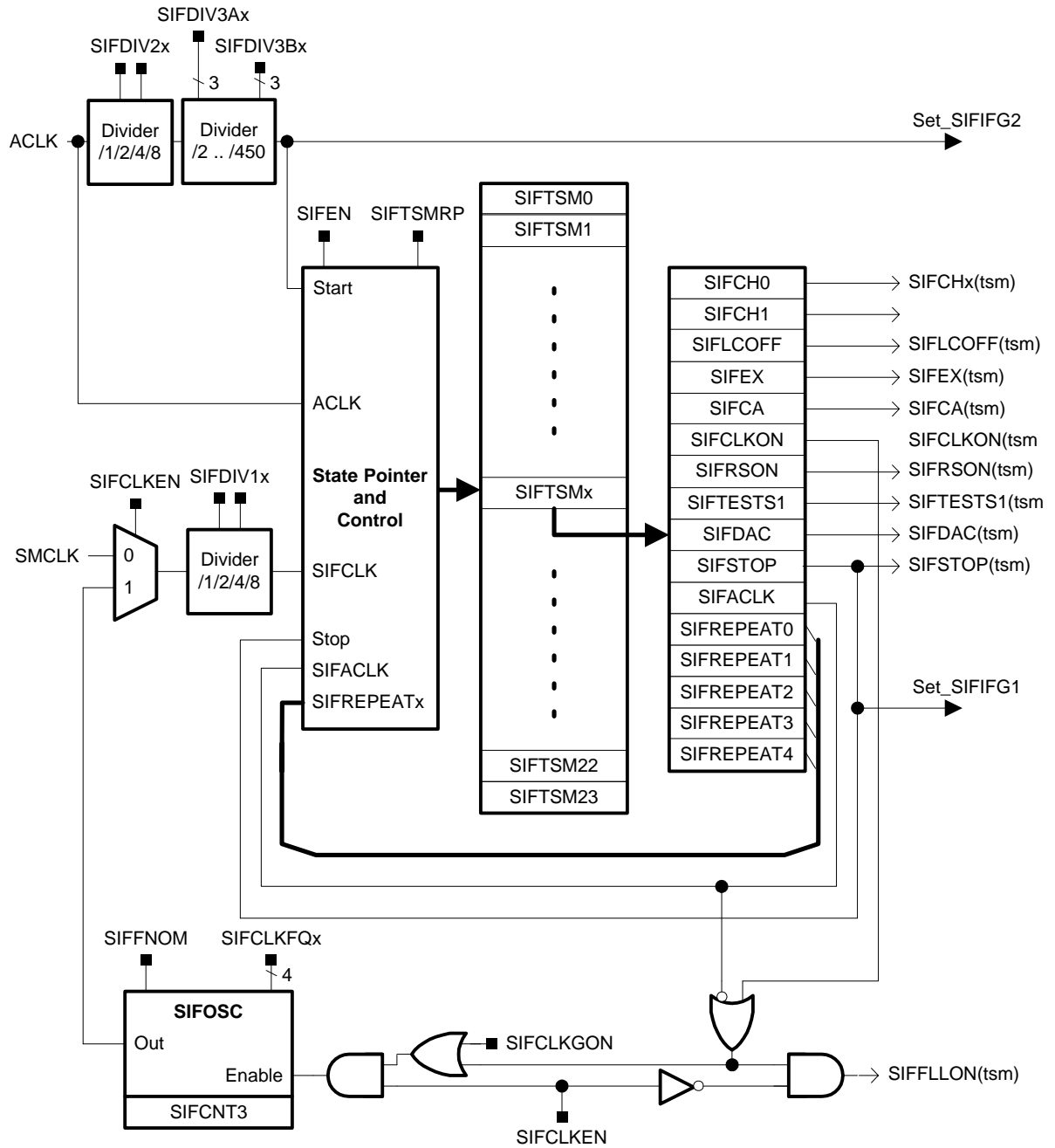
### 24.2.2 Scan IF Timing State Machine

The TSM is a sequential state machine that cycles through the SIFTSMx registers and controls the analog front end and sensor excitation automatically with no CPU intervention. The states are defined within a 24 x 16-bit memory, SIFTSM0 to SIFTSM23. The SIFEN bit enables the TSM. When SIFEN = 0, the ACLK input divider, the TSM start flip-flop, and the TSM outputs are reset and the internal oscillator is stopped. The TSM block diagram is shown in Figure 24–8.

The TSM begins at SIFTSM0 and ends when the TSM encounters a SIFTSMx state with a set SIFTSTOP bit. When a state with a set SIFTSTOP bit is reached, the state counter is reset to zero and state processing stops. State processing re-starts at SIFTSM0 with the next start condition when SIFTSMRP = 0, or immediately when SIFTSMRP = 1

After generation of the SIFSTOP(tsm) pulse, the timing state machine will load and maintain the conditions defined in SIFTSM0. In this state SIFLCEN(tsm) should be reset to ensure that all LC oscillators are shorted.

Figure 24–8. Timing State Machine Block Diagram



## TSM Operation

The TSM state machine automatically starts and re-starts periodically based on a divided ACLK start signal selected with the SIFDIV2x bits the SIFDIV3Ax and SIFDIV3Bx bits when SIFTSMRP = 0. For example, if SIFDIV3A and SIFDIV3B are configured to 270 ACLK cycles, then the TSM automatically starts every 270 ACLK cycles. When SIFTSMRP = 1 the TSM starts immediately with the SIFSTM0 state with the next ACLK cycle after encountering a state with a set SIFSTOP bit. The SIFIFG2 interrupt flag is set when the TSM starts.

The SIFDIV3Ax and SIFDIV3Bx bits may be updated anytime during operation. When updated, the current TSM sequence will continue with the old settings until the last state of the sequence completes. The new settings will take affect at the start of the next sequence.

## TSM Control of the AFE

The TSM controls the AFE with the SIFCHx, SIFLCEN, SIFEX, SIFCA, SIFRSON, SIFTESTS1, SIFDAC, and SIFSTOP bits. When any of these bits are set, their corresponding signal(s), SIFCHx(tsm), SIFLCEN(tsm), SIFEX(tsm), SIFCA(tsm), SIFRSON(tsm), SIFTESTS1(tsm), SIFDAC(tsm), and SIFSTOP(tsm) are high for the duration of the state. Otherwise, the corresponding signal(s) are low.

## TSM State Duration

The duration of each state is individually configurable with the SIFREPEATx bits. The duration of each state is SIFREPEATx + 1 times the selected clock source. For example, if a state were defined with SIFREPEATx = 3 and SIFACLK = 1, the duration of that state would be 4 x ACLK cycles. Because of clock synchronization, the duration of each state is affected by the clock source for the previous state, as shown in Table 24–5.

Table 24–5. TSM State Duration

SIFACLK		State Duration, T
For Previous State	For Current State	
0	0	$T = (\text{SIFREPEAT}_x + 1) \times 1/f_{\text{SIFCLK}}$
0	1	$(\text{SIFREPEAT}_x) \times 1/f_{\text{ACLK}} < T \leq (\text{SIFREPEAT}_x + 1) \times 1/f_{\text{ACLK}}$
1	0	$(\text{SIFREPEAT}_x + 1) \times 1/f_{\text{SIFCLK}} \leq T < (\text{SIFREPEAT}_x + 2) \times 1/f_{\text{SIFCLK}}$
1	1	$T = (\text{SIFREPEAT}_x + 1) \times 1/f_{\text{ACLK}}$

## TSM State Clock Source Select

The TSM clock source is individually configurable for each state. The TSM can be clocked from ACLK or a high frequency clock selected with the SIFACLK bit. When SIFACLK = 1, ACLK is used for the state, and when SIFACLK = 0, the high frequency clock is used. The high frequency clock can be sourced from SMCLK or the TSM internal oscillator, selected by the SIFCLKEN bit. The high-frequency clock can be divided by 1,2,4 or 8 with SIFDIV1x bits.

A set SIFCLKON bit is used to turn on the selected high frequency clock source for the duration of the state, when it is not used for the state. If the DCO is selected as the high frequency clock source, it is automatically turned on, regardless of the low-power mode settings of the MSP430.

The TSM internal oscillator generates a nominal frequency of 1MHz or 4MHz selected by the SIFFNOM bit and can be tuned in nominal 5% steps from -40% to +35% with the SIFCLKFQx. The frequency and the steps differ from device-to-device. See the device-specific datasheet for parameters.

The TSM internal oscillator frequency can be measured with ACLK. When SIFCLKEN = 1 and SIFCLKGON = 1 SIFCNT3 is reset, and beginning with the next rising edge of ACLK, SIFCNT3 counts the clock cycles of the internal oscillator. SIFCNT3 counts the internal oscillator cycles for one ACLK period when SIFNOM = 0 and four ACLK periods when SIFNOM = 1. Reading SIFCNT3 while it is counting will result in reading 01h.

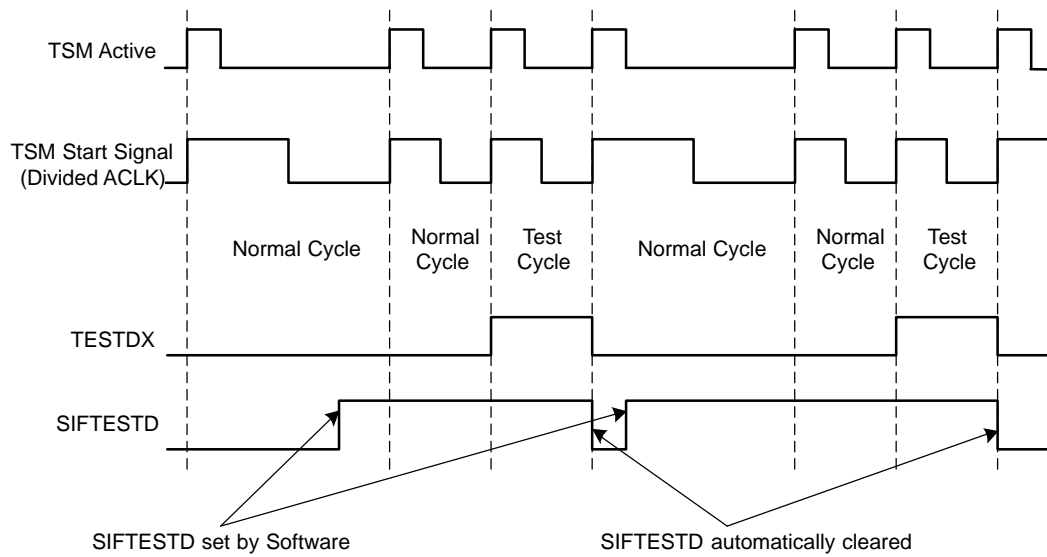
## TSM Stop Condition

The last state the TSM is marked with SIFSTOP = 1. The duration of this last state is always one SIFCLK cycle regardless of the SIFACLK or SIFREPEATx settings. The SIFIFG1 interrupt flag is set at when the TSM encounters a state with a set SIFSTOP bit.

## TSM Test Cycles

For calibration purposes, to detect sensor drift, or to measure signals other than the sensor signals, a test cycle may be inserted between TSM cycles by setting the SIFTESTD bit. The time between the TSM cycles is not altered by the test cycle insertion as shown in Figure 24–9. At the end of the test cycle the SIFTESTD bit is automatically cleared. The TESTDX signal is active during the test cycle to control input and output channel selection. TESTDX is generated after the SIFTESTD bit is set and the next TSM sequence completes.

Figure 24–9. Test Cycle Insertion



## TSM Example

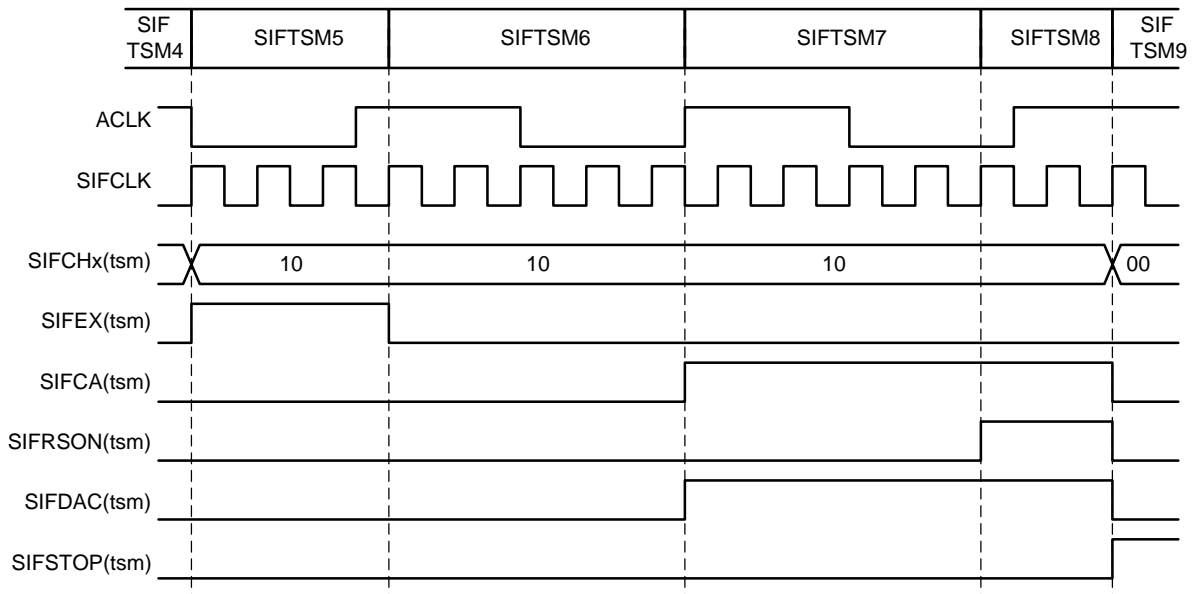
Figure 24–10 shows an example for a TSM sequence. The TSMx register values for the example are shown in Table 24–6. ACLK and SIFCLK are not drawn to scale. The TSM sequence starts with SIFTSM0 and ends with a set SIFSTOP bit in SIFTSM9. Only the SIFTSM5 to SIFTSM9 states are shown.

Table 24–6. TSM Example Register Values

TSMx Register	TSMx Register Contents
SIFTSM5	0100Ah
SIFTSM6	00402h
SIFTSM7	01812h
SIFTSM8	00952h
SIFTSM9	00200h

The example also shows the affects of the clock synchronization when switching between SIFCLK and ACLK. In state SIFTSM6, SIFACLK is set, whereas in the previous state and the successive state, SIFACLK is cleared. The waveform shows the duration of SIFTSM6 is less than one ACLK cycle and the duration of state SIFTSM7 is up to one SIFCLK period longer than configured by the SIFREPEATx bits.

Figure 24–10. Timing State Machine Example

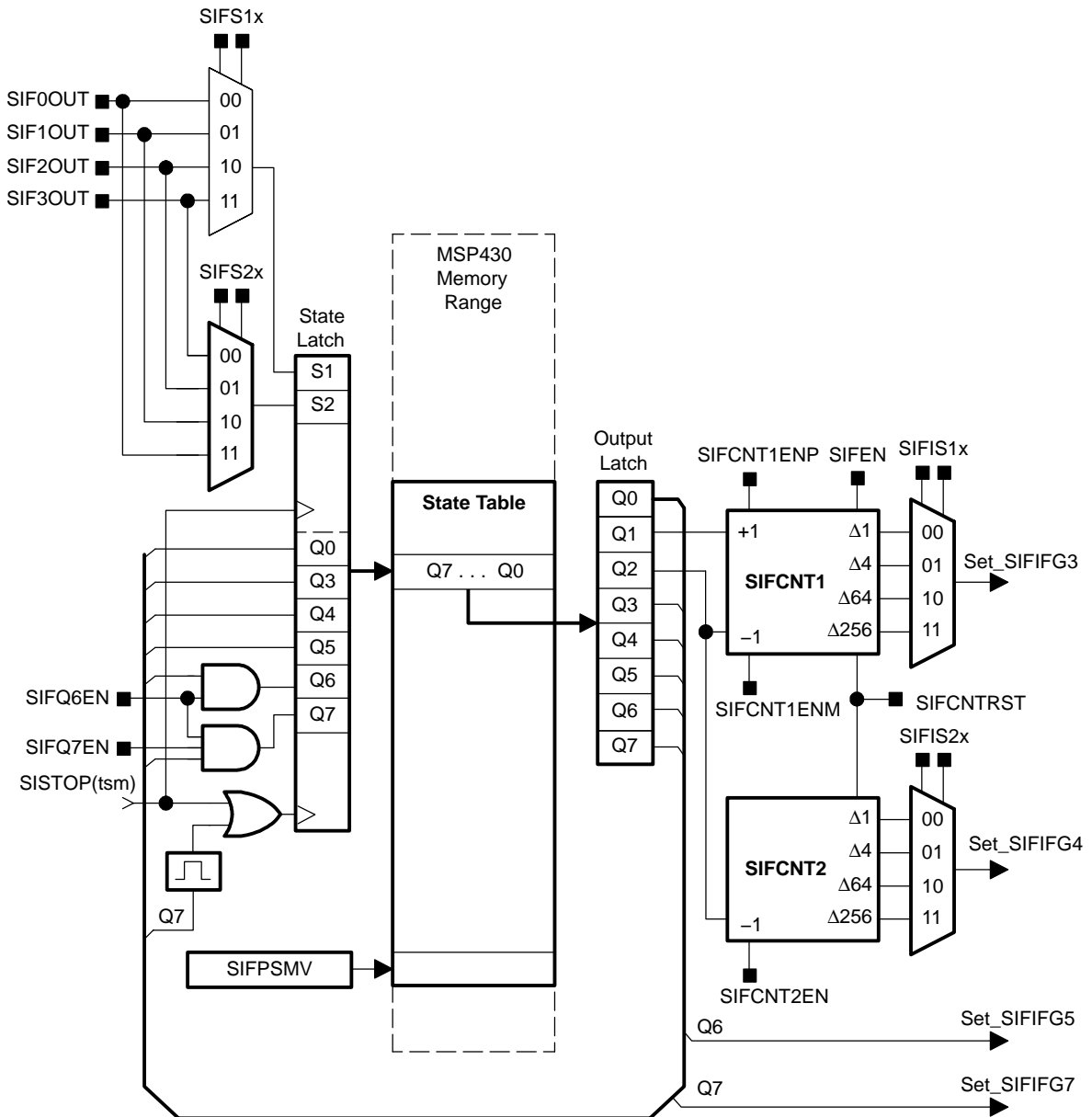


### 24.2.3 Scan IF Processing State Machine

The PSM is a programmable state machine used to determine rotation and direction with its state table stored within MSP430 memory (flash, ROM, or RAM). The processing state machine measures rotation and controls interrupt generation based on the inputs from the timing state machine and the analog front-end. The PSM vector SIFPSMV must to be initialized to point to the PSM state table. Multiple state tables are possible by reconfiguring the SIFPSMV to different tables as needed. The PSM block diagram is shown in Figure 24–11.



Figure 24–11. Scan IF Processing State Machine Block Diagram



### PSM Operation

At the falling edge of the SIFSTOP(tsm) signal the PSM moves the current-state byte from the PSM state table to the PSM output latch. The PSM has one dedicated channel of direct memory access (DMA), so all accesses to the PSM state table(s) are done automatically with no CPU intervention.

The current-state and next-state logic are reset while the Scan IF is disabled. One of the bytes stored at addresses SIFPSMV to SIFPSMV + 3 will be loaded first depending on the S1 and S2 signals when the Scan IF is enabled.

Signals S1 and S2 form a 2-bit offset added to the SIFPSMV contents to determine the first byte loaded to the PSM output latch. For example, when S2 = 1, and S1 = 0, the first byte loaded by the PSM will be at the address SIFPSMV + 2. The next byte and further subsequent bytes are determined by the next state calculations and are calculated by the PSM based on the state table contents and the values of signals S1 and S2.

**Note: SIFSTOP(tsm) Signal Frequency**

The SIFSTOP(tsm) signal frequency must be at least a factor of 32 lower than the MCLK. Otherwise, unpredictable operation could occur.

**Next State Calculation**

Bits 0, and 3 - 5 (Q0, Q3, Q4, Q5), and, if enabled by SIFQ6EN and SIFQ7EN, bits 6 and 7 (Q6, Q7) are used together with the signals S1 and S2 to calculate the next state. When SIFQ6EN = 1, Q6 is used in the next-state calculation. When SIFQ6EN = 1 and SIFQ7EN = 1, Q7 is used in the next-state calculation. The next state is:

Q7	Q6	Q5	Q4	Q3	Q0	S2	S1
----	----	----	----	----	----	----	----

When Q7 = 0, the PSM state is updated by the falling edge of the SIFSTOP(tsm) at the end of a TSM sequence. After updating the current state the PSM moves the corresponding state table entry to the output latch. When Q7 = 1, the next state is calculated immediately without waiting for the next falling edge of SIFSTOP(tsm), regardless of the state of SIFQ6EN or SIFQ7EN. The state is then updated with the next instruction fetch. The worst-case time between state transitions in this case is 6 MCLK cycles.

## PSM Counters

The PSM has two 8-bit counters SIFCNT1 and SIFCNT2. SIFCNT1 is updated with Q1 and Q2 and SIFCNT2 is updated with Q2. The counters can be read via the SIFCNT register. If the SIFCNTRST bit is set, each read access will reset the counters, otherwise the counters remain unchanged when read. If a count event occurs during a read access the count is postponed until the end of the read access but multiple count events during a read access will increment the counters only once. When SIFEN = 0, both counters are held in reset.

SIFCNT1 can increment or decrement based on Q1 and Q2. When SIFCNT1ENM = 1, SIFCNT1 decrements on a transition to a state where bit Q2 is set. When SIFCNT1ENP = 1, SIFCNT1 increments on a transition to a state where bit Q1 is set. When both bits SIFCNT1ENM and SIFCNT1ENP are set, and both bits Q1 and Q2 are set on a state transition, SIFCNT1 does not increment or decrement.

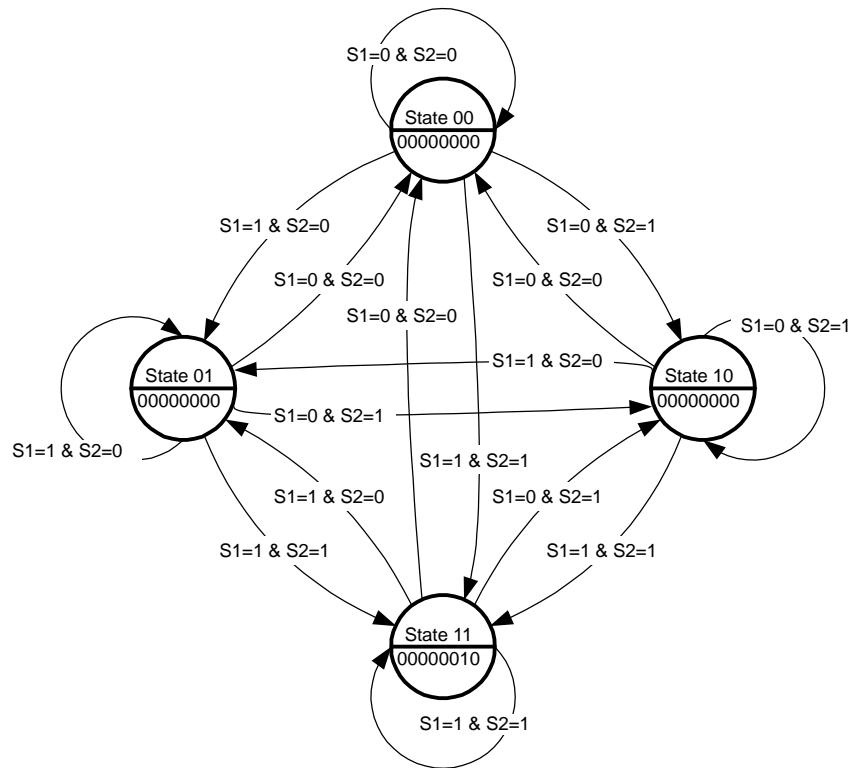
SIFCNT2 decrements based on Q2. When SIFCNT2EN = 1, SIFCNT2 decrements on a transition to a state where bit Q2 is set. On the first count after a reset SIFCNT2 will roll over from zero to 255 (0FFh).

When the next state is calculated to be the same state as the current state, the counters SIFCNT1 and SIFCNT2 are incremented or decremented according to Q1 and Q2 at the state transition. For example, if the current state is 05h and Q2 is set, and if the next state is calculated to be 05h, the transition from state 05h to 05h will decrement SIFCNT2 if SIFCNT2EN = 1.

## Simplest State Machine

Figure 24–12 shows the simplest state machine that can be realized with the PSM. The following code shows the corresponding state table and the PSM initialization.

Figure 24–12. Simplest PSM State Diagram



```

; Simplest State Machine Example
SIMPLEST_PSM db 000h      ; State 00 (State Table Index 0)
               db 000h      ; State 01 (State Table Index 1)
               db 000h      ; State 10 (State Table Index: 2)
               db 002h      ; State 11 (State Table Index: 3)

PSM_INIT
MOV   #SIMPLEST_PSM,&SIFPSMV   ; Init PSM vector
MOV   #SIFS20,&SIFCTL3         ; S1/S2 source
MOV   #SIFCNT1ENP+SIFCNT1ENM,&SIFCTL4
      ; Q7 and Q6 disabled for next state calc.
      ; Increment and decrement of SIFCNT1 enabled

```

If the PSM is in state 01 of the simplest state machine and the PSM has loaded the corresponding byte at index 01h of the state table:

Q7	Q6	Q5	Q4	Q3	Q2	Q1	Q0
0	0	0	0	0	0	0	0

For this example, S1 and S2 are set at the end of the next TSM sequence. To calculate the next state the bits Q5 - Q3 and Q0 of the state 01 table entry, together with the S1 and S2 signals are combined to form the next state:

Q7	Q6	Q5	Q4	Q3	Q0	S2	S1
0	0	0	0	0	0	1	1

The state table entry for state 11 is loaded at the next state transition:

Q7	Q6	Q5	Q4	Q3	Q2	Q1	Q0
0	0	0	0	0	0	1	0

Q1 is set in state 11, so SIFCNT1 will be incremented.

More complex state machines can be built by combining simple state machines to meet the requirements of specific applications.

#### 24.2.4 Scan IF Debug Register

The Scan IF peripheral has a SIFDEBUG register for debugging and development. Only the lower two bits should be written when writing to the SIFDEBUG register and only `MOV` instructions should be used write to SIFDEBUG. After writing the lower two bits, reading the SIFDEBUG contents gives the user different information. After writing 00h to SIFDEBUG, reading SIFDEBUG shows the last address read by the PSM. After writing 01h to SIFDEBUG, reading SIFDEBUG shows the index of the TSM and the PSM bits Q7 - Q0. After writing 02h to SIFDEBUG, reading SIFDEBUG shows the TSM output. After writing 03h to SIFDEBUG, reading SIFDEBUG shows which DAC register is selected and its contents.

### 24.2.5 Scan IF Interrupts

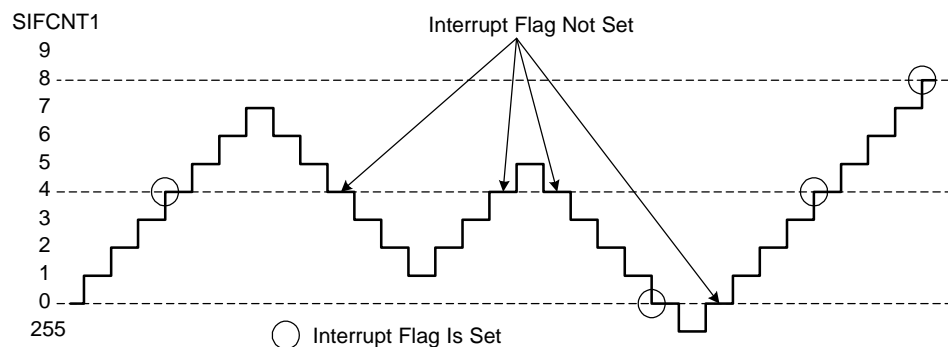
The Scan IF has one interrupt vector for seven interrupt flags listed in Table 24–7. Each interrupt flag has its own interrupt enable bit. When an interrupt is enabled, and the GIE bit is set, the interrupt flag will generate an interrupt. The interrupt flags are not automatically cleared. They must be cleared with software.

Table 24–7. Scan IF Interrupts

Interrupt Flag	Interrupt Condition
SIFIFG0	SIFIFG0 is set by one of the AFE SIFxOUT outputs selected with the SIFIFGSETx bits.
SIFIFG1	SIFIFG1 is set by the rising edge of the SIFSTOP(tsm) signal.
SIFIFG2	SIFIFG2 is set at the start of a TSM sequence.
SIFIFG3	SIFIFG3 is set at different count intervals of the SIFCNT1 counter, selected with the SIFIS1x bits.
SIFIFG4	SIFIFG4 is set at different count intervals of the SIFCNT2 counter, selected with the SIFIS2x bits.
SIFIFG5	SIFIFG5 is set when the PSM transitions to a state with Q6 set.
SIFIFG6	SIFIFG6 is set when the PSM transitions to a state with Q7 set.

Interrupt flags SIFIFG3 and SIFIFG4 have hysteresis so that the interrupt flag is set only once if the counter oscillates around the interrupt level as shown in Figure 24–13.

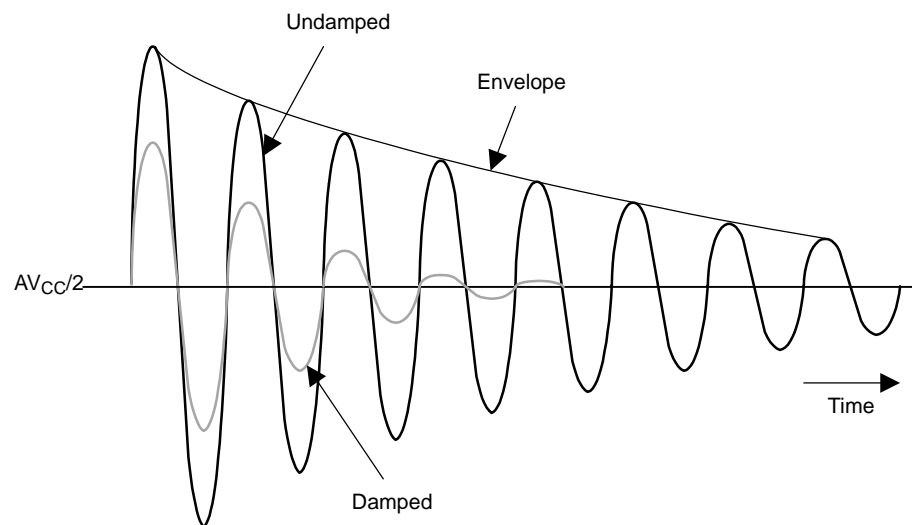
Figure 24–13. Interrupt Hysteresis Shown For Modulo 4 Interrupt Generation



### 24.2.6 Using the Scan IF with LC Sensors

Systems with LC sensors use a disk that is partially covered with a damping material to measure rotation. Rotation is measured with LC sensors by exciting the sensors and observing the resulting oscillation. The oscillation is either damped or un-damped by the rotating disk. The oscillation is always decaying because of energy losses but it decays faster when the damping material on the disk is within the field of the LC sensor, as shown in Figure 24–14. The LC oscillations can be measured with the oscillation test or the envelope test.

Figure 24–14. LC Sensor Oscillations

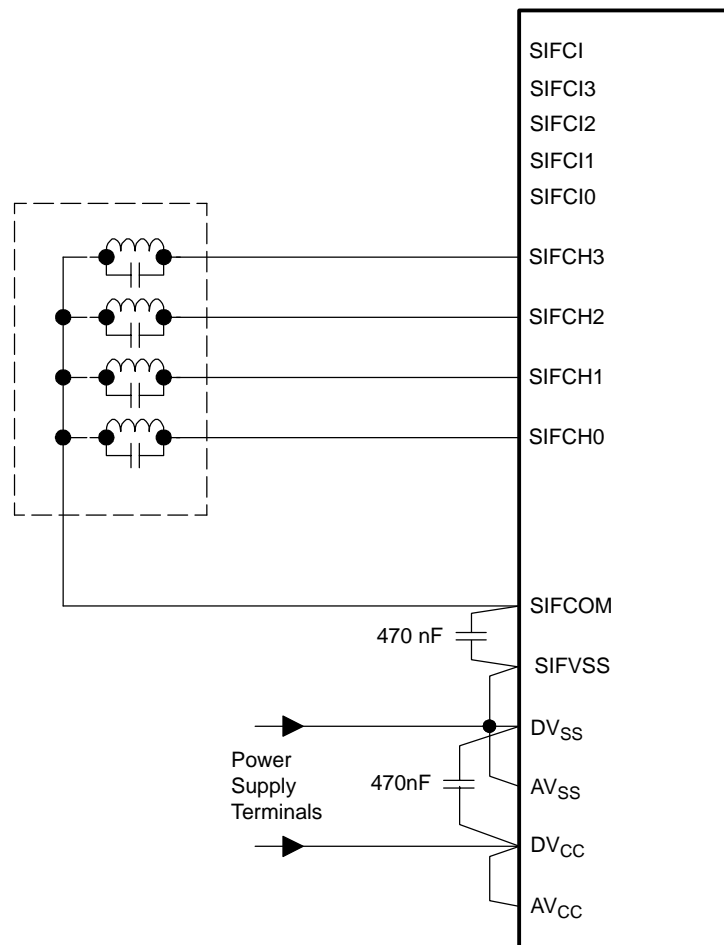




### 24.2.6.1 LC-Sensor Oscillation Test

The oscillation test tests if the amplitude of the oscillation after sensor excitation is above a reference level. The DAC is used to set the reference level for the comparator, and the comparator detects if the LC sensor oscillations are above or below the reference level. If the oscillations are above the reference level, the comparator will output a pulse train corresponding to the oscillations and the selected AFE output bit will 1. The measurement timing and reference level depend on the sensors and the system and should be chosen such that the difference between the damped and the un-damped amplitude is maximized. Figure 24–15 shows the connections for the oscillation test.

Figure 24–15. LC Sensor Connections For The Oscillation Test



### 24.2.6.2 LC-Sensor Envelope Test

The envelop test measures the decay time of the oscillations after sensor excitation. The oscillation envelope is created by the diodes and RC filters. The DAC is used to set the reference level for the comparator, and the comparator detects if the oscillation envelop is above or below the reference level. The comparator and AFE outputs are connected to Timer1\_A5 and the capture/compare registers for Timer1\_A5 are used to time the decay of the oscillation envelope. The PSM is not used for the envelope test.

When the sensors are connected to the individual SIFCIx inputs as shown in Figure 24–16, the comparator reference level can be adjusted for each sensor individually. When all sensors are connected to the SIFCI input as shown in Figure 24–17, only one comparator reference level is set for all sensors.

Figure 24–16. LC Sensor Connections For The Envelope Test

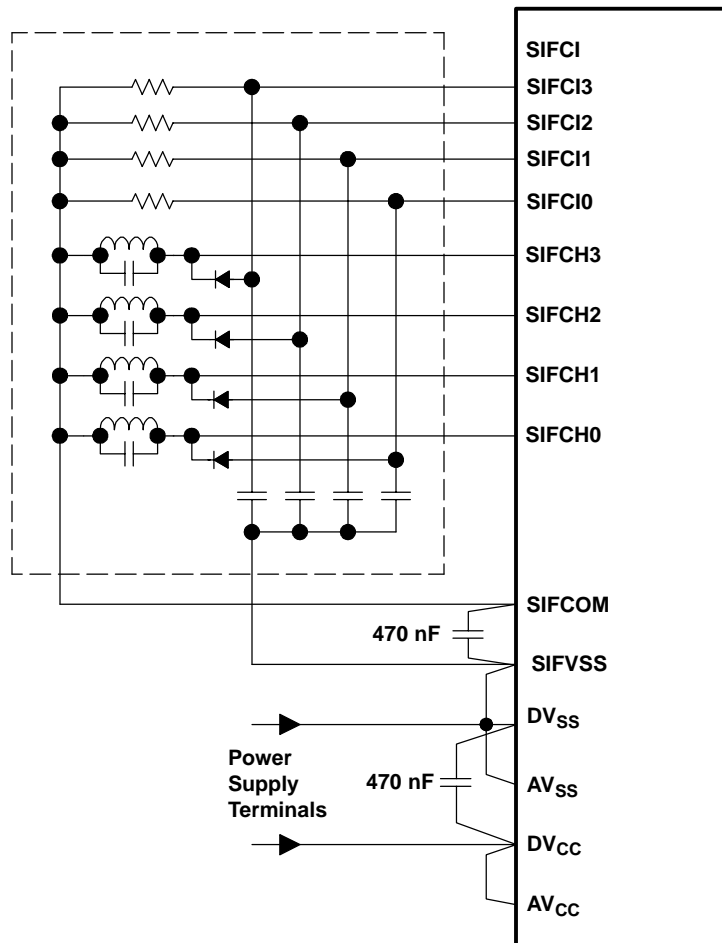
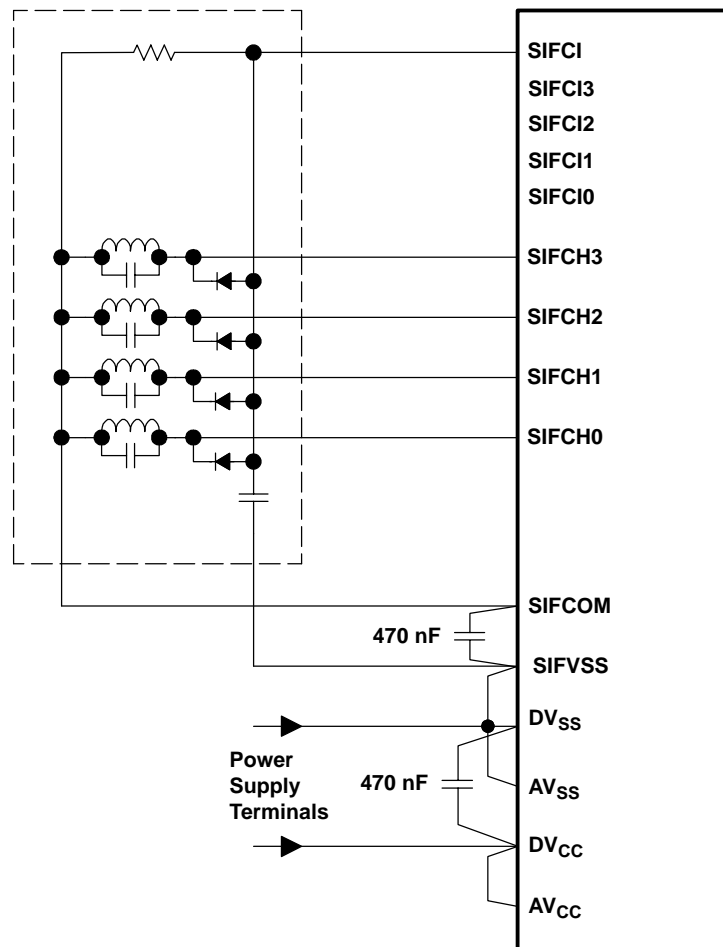


Figure 24-17. LC Sensor Connections For The Envelope Test

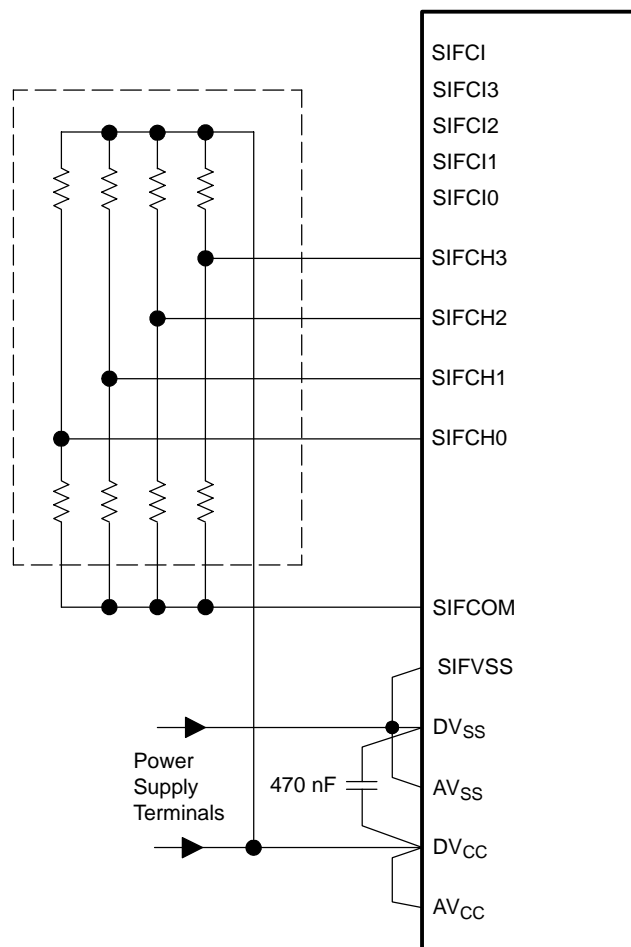


### 24.2.7 Using the Scan IF With Resistive Sensors

Systems with GMRs use magnets on an impeller to measure rotation. The damping material and magnets modify the electrical behavior of the sensor so that rotation and direction can be detected.

Rotation is measured with resistive sensors by connecting the resistor dividers to ground for a short time allowing current flow through the dividers. The resistors are affected by the rotating disc creating different divider voltages. The divider voltages are sampled with the sample-and-hold circuits. After the signals have settled the dividers may be switched off to prevent current flow and reduce power consumption. The DAC is used to set the reference level for the comparator, and the comparator detects if the sampled voltage is above or below the reference level. If the sampled voltage is above the reference level the comparator output is high. Figure 24–18 shows the connection for resistive sensors.

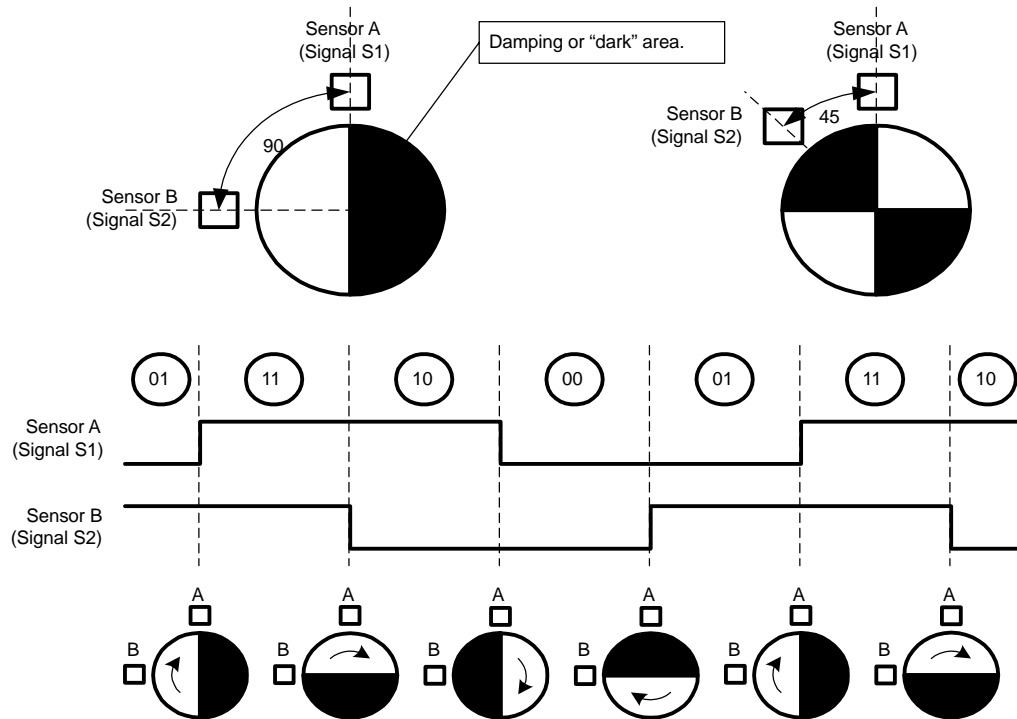
Figure 24–18. Resistive Sensor Connections



### 24.2.8 Quadrature Decoding

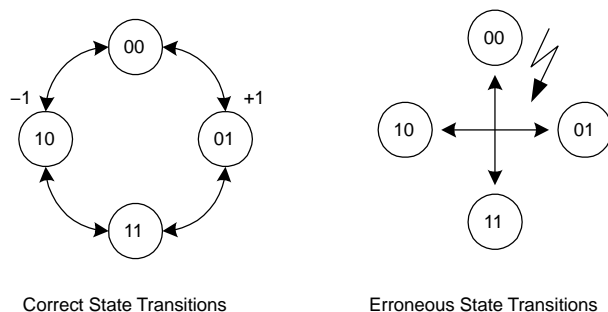
The Scan IF can be used to decode quadrature-encoded signals. Signals that are 90° out of phase with each other are said to be in quadrature. To create the signals, two sensors are positioned depending on the slotting, or coating of the encoder disk. Figure 24–19 shows two examples for the sensor positions and a quadrature-encoded signal waveform.

Figure 24–19. Sensor Position and Quadrature Signals



Quadrature decoding requires knowing the previous quadrature pair S1 and S2, as well as the current pair. Comparing these two pairs will tell the direction of the rotation. For example, if the current pair is 00 it can change to 01 or 10, depending on direction. Any other change in the signal pair would represent an error as shown in Figure 24–20.

Figure 24–20. Quadrature Decoding State Diagram



To transfer the state encoding into counts it is necessary to decide what fraction of the rotation should be counted and on what state transitions. In this example only full rotations will be counted on the transition from state 00 to 01 or 10 using a 180° disk with the sensors 90° apart. All the possible state transitions can be put into a table and this table can be translated into the corresponding state table entries for the processing state machine as shown in Table 24–8.

Table 24–8. Quadrature Decoding PSM Table

Previous Quad. Pair	Current Quad. Pair	Movement	State Table Entry					
			Q6	Q2	Q1	Q3	Q0	
			Error	-1	+1	Current Quad. Pair	Byte Code	
00	00	No Rotation	0	0	0	0	0	000h
00	01	Turns right, +1	0	0	1	0	1	003h
00	10	Turns left, -1	0	1	0	1	0	00Ch
00	11	Error	1	0	0	1	1	049h
01	00	Turns left	0	0	0	0	0	000h
01	01	No rotation	0	0	0	0	1	001h
01	10	Error	1	0	0	1	0	048h
01	11	Turns right	0	0	0	1	1	009h
10	00	Turns right	0	0	0	0	0	000h
10	01	Error	1	0	0	0	1	041h
10	10	No rotation	0	0	0	1	0	008h
10	11	Turns left	0	0	0	1	1	009h
11	00	Error	1	0	0	0	0	040h
11	01	Turns left	0	0	0	0	1	001h
11	10	Turns right	0	0	0	1	0	008h
11	11	No rotation	0	0	0	1	1	009h

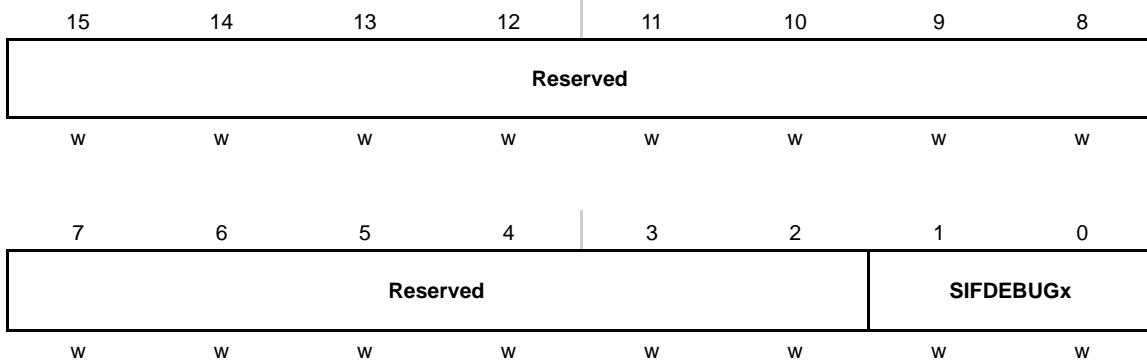
## 24.3 Scan IF Registers

The Scan IF registers are listed in Table 24–9.

Table 24–9. Scan IF Registers

Register	Short Form	Register Type	Address	Initial State
Scan IF debug register	SIFDEBUG	Read/write	01B0h	Unchanged
Scan IF counter 1 and 2	SIFCNT	Read/write	01B2h	Reset with POR
Scan IF PSM vector	SIFPSMV	Read/write	01B4h	Unchanged
Scan IF control 1	SIFCTL1	Read/write	01B6h	Reset with POR
Scan IF control 2	SIFCTL2	Read/write	01B8h	Reset with POR
Scan IF control 3	SIFCTL3	Read/write	01BAh	Reset with POR
Scan IF control 4	SIFCTL4	Read/write	01BCh	Reset with POR
Scan IF control 5	SIFCTL5	Read/write	01BEh	Reset with POR
Scan IF DAC 0	SIFDACR0	Read/write	01C0h	Unchanged
Scan IF DAC 1	SIFDACR1	Read/write	01C2h	Unchanged
Scan IF DAC 2	SIFDACR2	Read/write	01C4h	Unchanged
Scan IF DAC 3	SIFDACR3	Read/write	01C6h	Unchanged
Scan IF DAC 4	SIFDACR4	Read/write	01C8h	Unchanged
Scan IF DAC 5	SIFDACR5	Read/write	01CAh	Unchanged
Scan IF DAC 6	SIFDACR6	Read/write	01CCh	Unchanged
Scan IF DAC 7	SIFDACR7	Read/write	01CEh	Unchanged
Scan IF TSM 0	SIFTSM0	Read/write	01D0h	Unchanged
Scan IF TSM 1	SIFTSM1	Read/write	01D2h	Unchanged
Scan IF TSM 2	SIFTSM2	Read/write	01D4h	Unchanged
Scan IF TSM 3	SIFTSM3	Read/write	01D6h	Unchanged
Scan IF TSM 4	SIFTSM4	Read/write	01D8h	Unchanged
Scan IF TSM 5	SIFTSM5	Read/write	01DAh	Unchanged
Scan IF TSM 6	SIFTSM6	Read/write	01DCh	Unchanged
Scan IF TSM 7	SIFTSM7	Read/write	01DEh	Unchanged
Scan IF TSM 8	SIFTSM8	Read/write	01E0h	Unchanged
Scan IF TSM 9	SIFTSM9	Read/write	01E2h	Unchanged
Scan IF TSM 10	SIFTSM10	Read/write	01E4h	Unchanged
Scan IF TSM 11	SIFTSM11	Read/write	01E6h	Unchanged
Scan IF TSM 12	SIFTSM12	Read/write	01E8h	Unchanged
Scan IF TSM 13	SIFTSM13	Read/write	01EAh	Unchanged
Scan IF TSM 14	SIFTSM14	Read/write	01ECh	Unchanged
Scan IF TSM 15	SIFTSM15	Read/write	01EEh	Unchanged
Scan IF TSM 16	SIFTSM16	Read/write	01F0h	Unchanged
Scan IF TSM 17	SIFTSM17	Read/write	01F2h	Unchanged
Scan IF TSM 18	SIFTSM18	Read/write	01F4h	Unchanged
Scan IF TSM 19	SIFTSM19	Read/write	01F6h	Unchanged
Scan IF TSM 20	SIFTSM20	Read/write	01F8h	Unchanged
Scan IF TSM 21	SIFTSM21	Read/write	01FAh	Unchanged
Scan IF TSM 22	SIFTSM22	Read/write	01FCh	Unchanged
Scan IF TSM 23	SIFTSM23	Read/write	01FEh	Unchanged

**SIFDEBUG, Scan IF Debug Register, Write Mode**

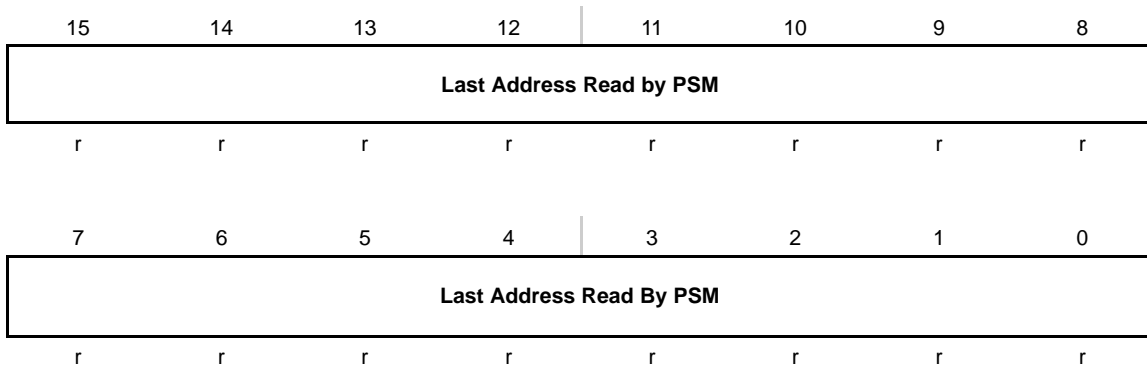


**Reserved** Bits 15-2 Reserved. Must be written as zero.

**SIFDEBUGx** Bits 1-0 SIFDEBUG register mode. Writing these bits selects the read-mode of the SIFDEBUG register. SIFDEBUG must be written with MOV instructions only.

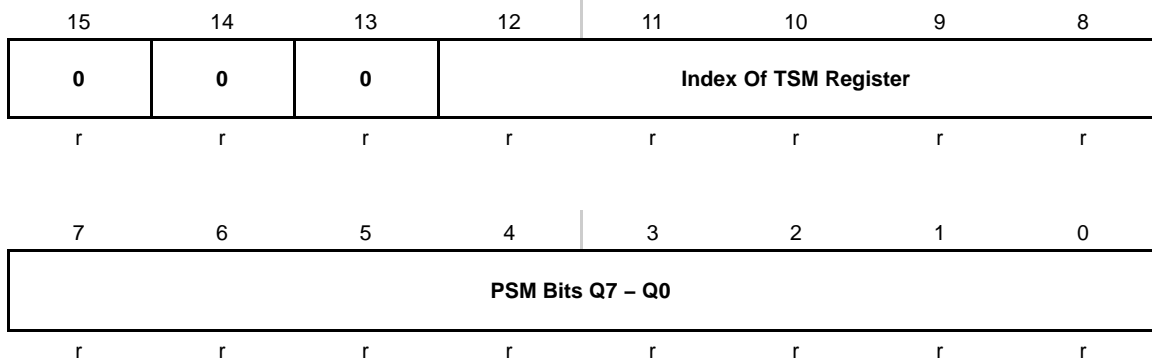
- 00 When read, SIFDEBUG shows the last address read by the PSM
- 01 When read, SIFDEBUG shows the value of the TSM state pointer and the PSM bits Q7 - Q0
- 10 When read, SIFDEBUG shows the contents of the current SIFTSMx register.
- 11 When read, SIFDEBUG shows the currently selected DAC register and its contents.

**SIFDEBUG, Scan IF Debug Register, Read Mode After 00h Is Written**

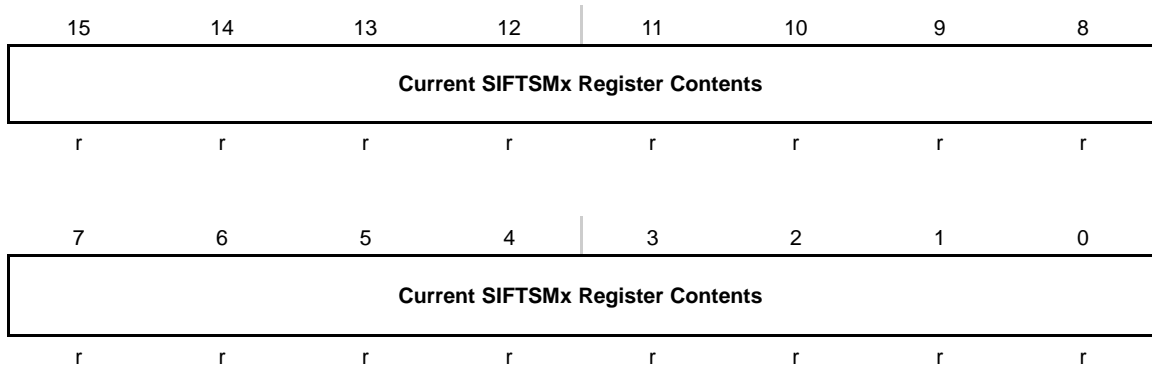


**Last PSM** Bits 15-0 When SIFDEBUG is read, after 00h has been written to it, SIFDEBUG shows the last address read by the PSM.



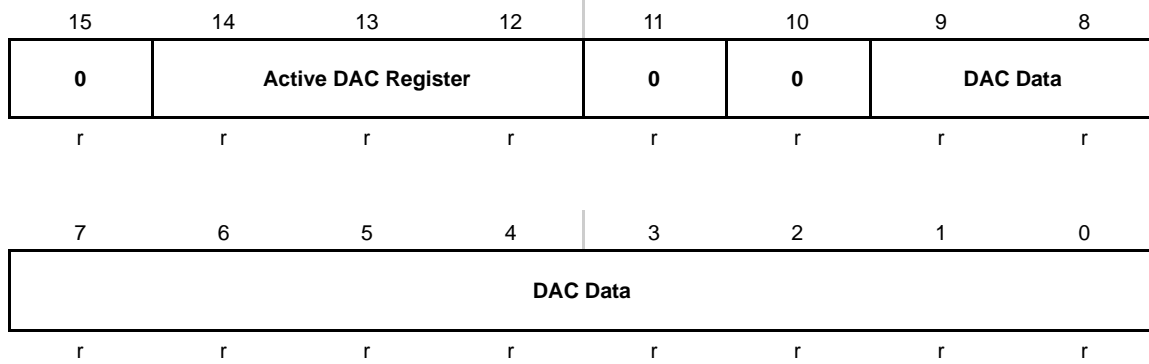
**SIFDEBUG, Scan IF Debug Register, Read Mode After 01h Is Written**

<b>Unused</b>	Bits 15-13	Unused. After 01h is written to SIFDEBUG, these bits are always read as zero.
<b>TSM Index</b>	Bits 12-8	When SIFDEBUG is read, after 01h is written to it, these bits show the TSM register pointer index.
<b>PSM Bits</b>	Bits 7-0	When SIFDEBUG is read, after 01h is written to it, these bits show the PSM bits Q7 - Q0.

**SIFDEBUG, Scan IF Debug Register, Read Mode After 02h Is Written**

Bits 15-0	When SIFDEBUG is read, after 02h is written to it, these bits show the TSM output.
--------------	--

**SIFDEBUG, Scan IF Debug Register, Read Mode After 03h Is Written**

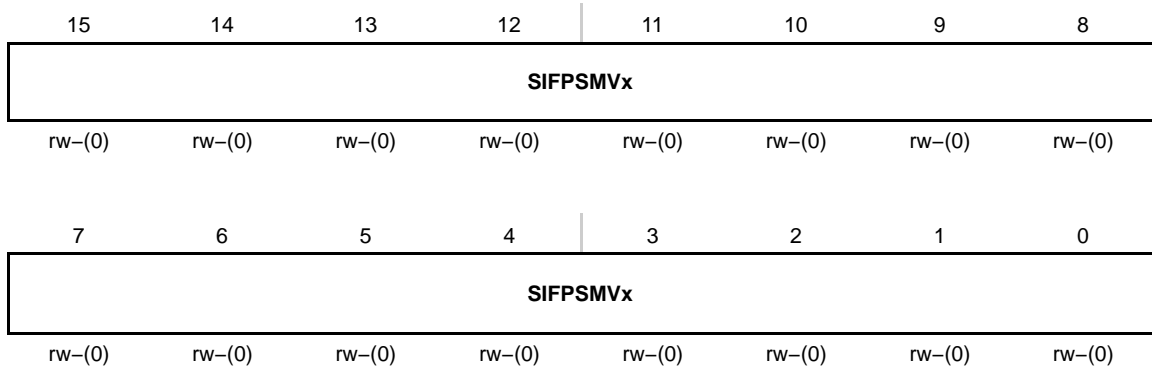


- |               |        |  |
|---------------|--------|--|
| <b>Unused</b> | Bit 15 | Unused. After 03h is written to SIFDEBUG, this bit is always read as zero. |
|---------------|--------|--|
- |                     |            |   |
|---------------------|------------|---|
| <b>DAC Register</b> | Bits 14-12 | When SIFDEBUG is read, after 03h is written to it, these bits show which DAC register is currently selected to control the DAC. |
|---------------------|------------|---|
- |               |            |   |
|---------------|------------|---|
| <b>Unused</b> | Bits 11-10 | Unused. After 03h is written to SIFDEBUG, these bits are always read as zero. |
|---------------|------------|---|
- |                 |          |  |
|-----------------|----------|--|
| <b>DAC Data</b> | Bits 9-0 | When SIFDEBUG is read, after 03h is written to it, these bits show value of the currently-selected DAC register. |
|-----------------|----------|--|

**SIFCNT, Scan IF Counter Register**

**SIFCNT2x** Bits 15-8 SIFCNT2. These bits are the SIFCNT2 counter. SIFCNT2 is reset when SIFEN = 0 or if read when SIFCNTRST = 1.

**SIFCNT1x** Bits 7-0 SIFCNT1. These bits are the SIFCNT1 counter. SIFCNT1 is reset when SIFEN = 0 or if read when SIFCNTRST = 1.

**SIFPSMV, Scan IF Processing State Machine Vector Register**

**SIFPSMVx** Bits 15-0 SIF PSM vector. These bits are the address for the first state in the PSM state table.

**SIFCTL1, Scan IF Control Register 1**

15	14	13	12	11	10	9	8
<b>SIFIE6</b>	<b>SIFIE5</b>	<b>SIFIE4</b>	<b>SIFIE3</b>	<b>SIFIE2</b>	<b>SIFIE1</b>	<b>SIFIE0</b>	<b>SIFIFG6</b>
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
<b>SIFIFG5</b>	<b>SIFIFG4</b>	<b>SIFIFG3</b>	<b>SIFIFG2</b>	<b>SIFIFG1</b>	<b>SIFIFG0</b>	<b>SIFTESTD</b>	<b>SIFEN</b>
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

<b>SIFIE<sub>x</sub></b>	Bits 15-9	Interrupt Enable. These bits enable or disable the interrupt request for the SIFIFG <sub>x</sub> bits. 0    Interrupt disabled 1    Interrupt enabled
<b>SIFIFG6</b>	Bit 8	SIF interrupt flag 6. This bit is set when the PSM transitions to a state with a set Q7 bit. SIFIFG6 must be reset with software. 0    No interrupt pending 1    Interrupt pending
<b>SIFIFG5</b>	Bit 7	SIF interrupt flag 5. This bit is set when the PSM transitions to a state with a set Q6 bit. SIFIFG5 must be reset with software. 0    No interrupt pending 1    Interrupt pending
<b>SIFIFG4</b>	Bit 6	SIF interrupt flag 4. This bit is set by the SIFCNT2 counter conditions selected with the SIFIS2 <sub>x</sub> bits. SIFIFG4 must be reset with software. 0    No interrupt pending 1    Interrupt pending
<b>SIFIFG3</b>	Bit 5	SIF interrupt flag 3. This bit is set by the SIFCNT1 counter conditions selected with the SIFIS1 <sub>x</sub> bits. SIFIFG3 must be reset with software. 0    No interrupt pending 1    Interrupt pending
<b>SIFIFG2</b>	Bit 4	SIF interrupt flag 2. This bit is set at the start of a TSM sequence. SIFIFG2 must be reset with software. 0    No interrupt pending 1    Interrupt pending
<b>SIFIFG1</b>	Bit 3	SIF interrupt flag 1. This bit is set by the rising edge of the SIFSTOP(tsm) signal. SIFIFG1 must be reset with software. 0    No interrupt pending 1    Interrupt pending

---

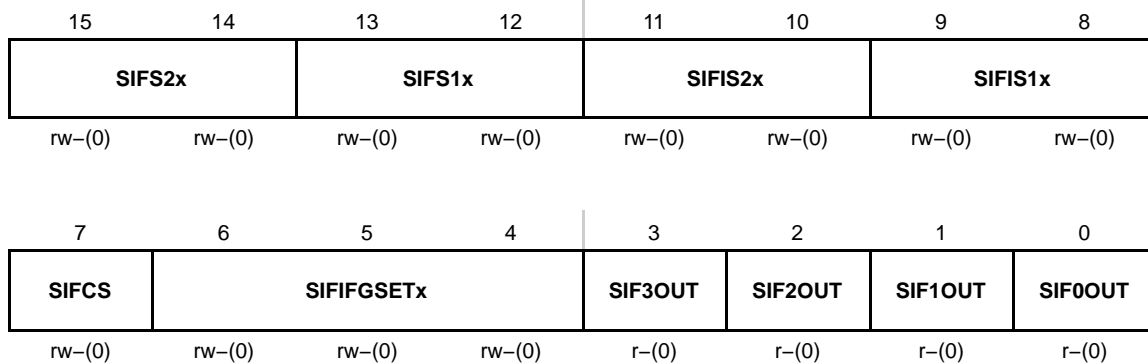
<b>SIFFG0</b>	Bit 2	SIF interrupt flag 0. This bit is set by the SIFxOUT conditions selected by the SIFIFGSETx bits. SIFFG0 must be reset with software. 0 No interrupt pending 1 Interrupt pending
<b>SIFTESTD</b>	Bit 1	Test cycle insertion. Setting this bit inserts a test cycle between TSM cycles. SIFTESTD is automatically reset at the end of the test cycle. 0 No test cycle inserted 1 Test cycle inserted between TSM cycles.
<b>SIFEN</b>	Bit 0	Scan interface enable. Setting this bit enables the Scan IF. 0 Scan IF disabled 1 Scan IF enabled

**SIFCTL2, Scan IF Control Register 2**

15	14	13	12	11	10	9	8
<b>SIFDACON</b>	<b>SIFCAON</b>	<b>SIFCAINV</b>	<b>SIFCAX</b>	<b>SIFCISEL</b>	<b>SIFCACI3</b>	<b>SIFVSS</b>	<b>SIFVCC2</b>
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
<b>SIFSH</b>	<b>SIFTEN</b>	<b>SIFTCH1x</b>		<b>SIFTCH0x</b>		<b>SIFTCH1 OUT</b>	<b>SIFTCH0 OUT</b>
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

- SIFDACON** Bit 15 DAC on. Setting this bit turns the DAC on regardless of the TSM control.  
0 The DAC is controlled by the TSM.  
1 The DAC is on.
- SIFCAON** Bit 14 Comparator on. Setting this bit turns the comparator on regardless of the TSM control.  
0 The comparator is controlled by the TSM.  
1 The comparator is on.
- SIFCAINV** Bit 13 Invert comparator output  
0 Comparator output is not inverted  
1 Comparator output is inverted
- SIFCAX** Bit 12 Comparator input select. This bit selects groups of signals for the comparator input.  
0 Comparator input is one of the SIFCHx channels, selected with the channel select logic.  
1 Comparator input is one of the SIFCIx channels, selected with the channel select logic and the SIFCISEL and SIFCACI3 bits.
- SIFCISEL** Bit 11 Comparator input select. This bit is used with the SIFCACI3 bit to select the comparator input when SIFCAX = 1.  
0 Comparator input is one of the SIFCIx channels, selected with the channel select logic and SIFCACI3 bit.  
1 Comparator input is the SIFCI channel
- SIFCACI3** Bit 10 Comparator input select. This bit is selects the comparator input when SIFCISEL = 0 and SIFCAX = 1.  
0 Comparator input is selected with the channel select logic.  
1 Comparator input is SIFCI3.
- SIFVSS** Bit 9 Sample-and-hold SIFV<sub>SS</sub> select.  
0 The ground connection of the sample capacitor is connected to SIFV<sub>SS</sub>, regardless of the TSM control.  
1 The ground connection of the sample capacitor is controlled by the TSM

<b>SIFVCC2</b>	Bit 8	Mid-voltage generator 0 $AV_{CC}/2$ generator is off 1 $AV_{CC}/2$ generator is on if SIFSH = 0
<b>SIFSH</b>	Bit 7	Sample-and-hold enable 0 Sample-and-hold is disabled 1 Sample-and-hold is enabled
<b>SIFTEN</b>	Bit 6	Excitation enable 0 Excitation circuitry is disabled 1 Excitation circuitry is enabled
<b>SIFTCH1x</b>	Bits 5-4	These bits select the comparator input for test channel 1. 00 Comparator input is SIFCH0 when SIFCAX = 0 Comparator input is SIFCI0 when SIFCAX = 1 01 Comparator input is SIFCH1 when SIFCAX = 0 Comparator input is SIFCI1 when SIFCAX = 1 10 Comparator input is SIFCH2 when SIFCAX = 0 Comparator input is SIFCI2 when SIFCAX = 1 11 Comparator input is SIFCH3 when SIFCAX = 0 Comparator input is SIFCI3 when SIFCAX = 1
<b>SIFTCH0x</b>	Bits 3-2	These bits select the comparator input for test channel 0. 00 Comparator input is SIFCH0 when SIFCAX = 0 Comparator input is SIFCI0 when SIFCAX = 1 01 Comparator input is SIFCH1 when SIFCAX = 0 Comparator input is SIFCI1 when SIFCAX = 1 10 Comparator input is SIFCH2 when SIFCAX = 0 Comparator input is SIFCI2 when SIFCAX = 1 11 Comparator input is SIFCH3 when SIFCAX = 0 Comparator input is SIFCI3 when SIFCAX = 1
<b>SIFTCH1 OUT</b>	Bit 1	AFE output for test channel 1
<b>SIFTCH0 OUT</b>	Bit 0	AFE output for test channel 0

**SIFCTL3, Scan IF Control Register 3**

<b>SIFS2x</b>	Bits 15-14	<p>S2 source select. These bits select the S2 source for the PSM when SIFCS = 1.</p> <p>00 SIF0OUT is the S2 source. 01 SIF1OUT is the S2 source. 10 SIF2OUT is the S2 source. 11 SIF3OUT is the S2 source.</p>
<b>SIFS1x</b>	Bits 13-12	<p>S1 source select. These bits select the S1 source fro the PSM when SIFCS = 1.</p> <p>00 SIF0OUT is the S1 source. 01 SIF1OUT is the S1 source. 10 SIF2OUT is the S1 source. 11 SIF3OUT is the S1 source.</p>
<b>SIFIS2x</b>	Bits 11-10	<p>SIFIFG4 interrupt flag source</p> <p>00 SIFIFG4 is set with each count of SIFCNT2. 01 SIFIFG4 is set if (SIFCNT2 modulo 4) = 0. 10 SIFIFG4 is set if (SIFCNT2 modulo 64) = 0. 11 SIFIFG4 is set when SIFCNT2 decrements from 01h to 00h.</p>
<b>SIFIS1x</b>	Bits 9-8	<p>SIFIFG3 interrupt flag source</p> <p>00 SIFIFG3 is set with each count, up or down, of SIFCNT1. 01 SIFIFG3 is set if (SIFCNT1 modulo 4) = 0. 10 SIFIFG3 is set if (SIFCNT1 modulo 64) = 0. 11 SIFIFG3 is set when SIFCNT1 rolls over from 0FFh to 00h.</p>
<b>SIFCS</b>	Bit 7	<p>Comparator output/Timer_A input selection</p> <p>0 The SIFEX(tsm) signal and the comparator output are connected to the TACCRx inputs. 1 The SIFxOUT outputs are connected to the TACCRx inputs selected with the SIFS1x and SIFS2x bits.</p>



---

<b>SIFIFGSETx</b>	Bits 6-4	SIFIFG0 interrupt flag source. These bits select when the SIFIFG0 flag is set. 000 SIFIFG0 is set when SIF0OUT is set. 001 SIFIFG0 is set when SIF0OUT is reset. 010 SIFIFG0 is set when SIF1OUT is set. 011 SIFIFG0 is set when SIF1OUT is reset. 100 SIFIFG0 is set when SIF2OUT is set. 101 SIFIFG0 is set when SIF2OUT is reset. 110 SIFIFG0 is set when SIF3OUT is set. 111 SIFIFG0 is set when SIF3OUT is reset.
<b>SIF3OUT</b>	Bit 3	AFE output bit 3
<b>SIF2OUT</b>	Bit 2	AFE output bit 2
<b>SIF1OUT</b>	Bit 1	AFE output bit 1
<b>SIF0OUT</b>	Bit 0	AFE output bit 0

**SIFCTL4, Scan IF Control Register 4**

15	14	13	12	11	10	9	8
<b>SIFCNTRST</b>	<b>SIFCNT2EN</b>	<b>SIFCNT1 ENM</b>	<b>SIFCNT1 ENP</b>	<b>SIFQ7EN</b>	<b>SIFQ6EN</b>	<b>SIFDIV3Bx</b>	
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
<b>SIFDIV3Bx</b>	<b>SIFDIV3Ax</b>			<b>SIFDIV2x</b>		<b>SIFDIV1x</b>	
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

- SIFCNTRST** Bit 15 Counter reset. Setting this bit enables the SIFCNT register to be reset when it is read.  
0 SIFCNT register is not reset when read  
1 SIFCNT register is reset when it is read
- SIFCNT2EN** Bit 14 SIFCNT2 enable  
0 SIFCNT2 is disabled  
1 SIFCNT2 is enabled
- SIFCNT1 ENM** Bit 13 SIFCNT1 decrement enable  
0 SIFCNT1 decrement is disabled  
1 SIFCNT1 decrement is enabled
- SIFCNT1 ENP** Bit 12 SIFCNT1 increment enable  
0 SIFCNT1 increment is disabled  
1 SIFCNT1 increment is enabled
- SIFQ7EN** Bit 11 Q7 enable. This bit enables bit Q7 for the next PSM state calculation when SIFQ6EN = 1.  
0 Q7 is not used to determine the next PSM state  
1 Q7 is used to determine the next PSM state
- SIFQ6EN** Bit 10 Q6 enable. This bit enables Q6 for the next PSM state calculation.  
0 Q6 is not used to determine the next PSM state  
1 Q6 is used to determine the next PSM state

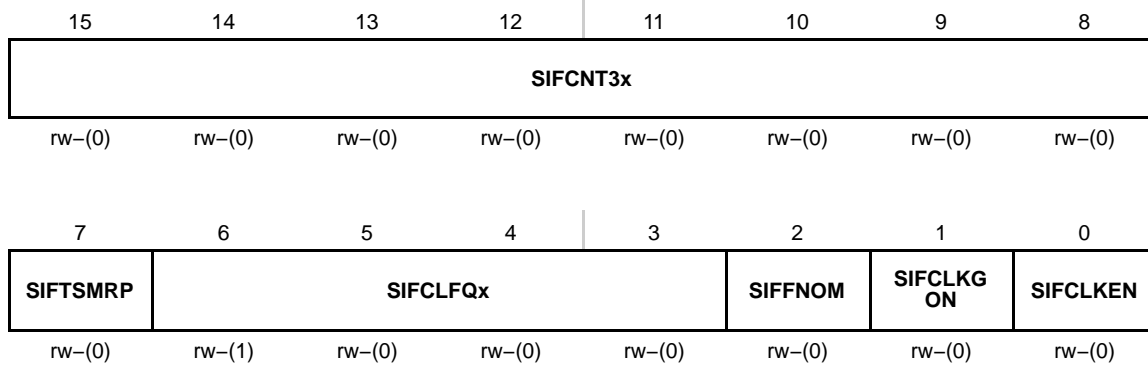
**SIFDIV3Bx** Bits 9-7 TSM start trigger ACLK divider. These bits together with the SIFDIV3Ax bits select the division rate for the TSM start trigger.

**SIFDIV3Ax** Bits 6-4 TSM start trigger ACLK divider. These bits together with the SIFDIV3Bx bits select the division rate for the TSM start trigger. The division rate is:

SIFDIV3Bx	SIFDIV3Ax							
	000	001	010	011	100	101	110	111
000	2	6	10	14	18	22	26	30
001	6	18	30	42	54	66	78	90
010	10	30	50	70	90	110	130	150
011	14	42	70	98	126	154	182	210
100	18	54	90	126	162	198	234	270
101	22	66	110	154	198	242	286	330
110	26	78	130	182	234	286	338	390
111	30	90	150	210	270	330	390	450

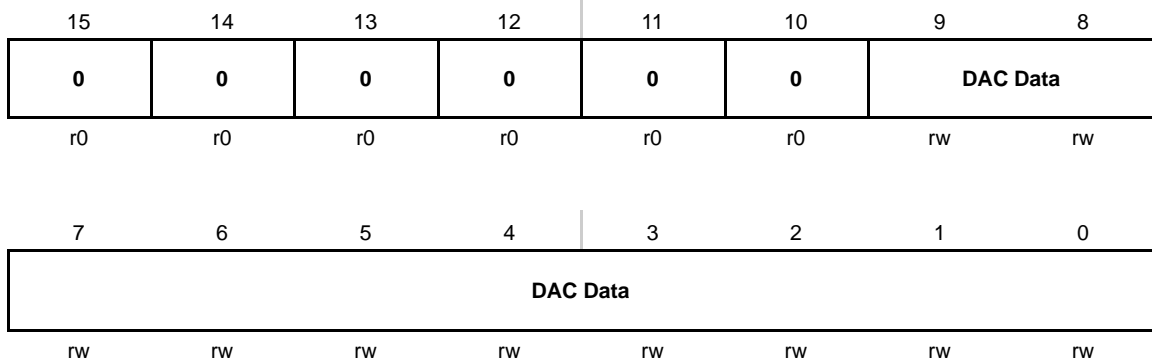
**SIFDIV2x** Bits 3-2 ACLK divider. These bits select the ACLK division.  
 00 /1  
 01 /2  
 10 /4  
 11 /8

**SIFDIV1x** Bits 1-0 TSM SMCLK divider. These bits select the SMCLK division for the TSM.  
 00 /1  
 01 /2  
 10 /4  
 11 /8

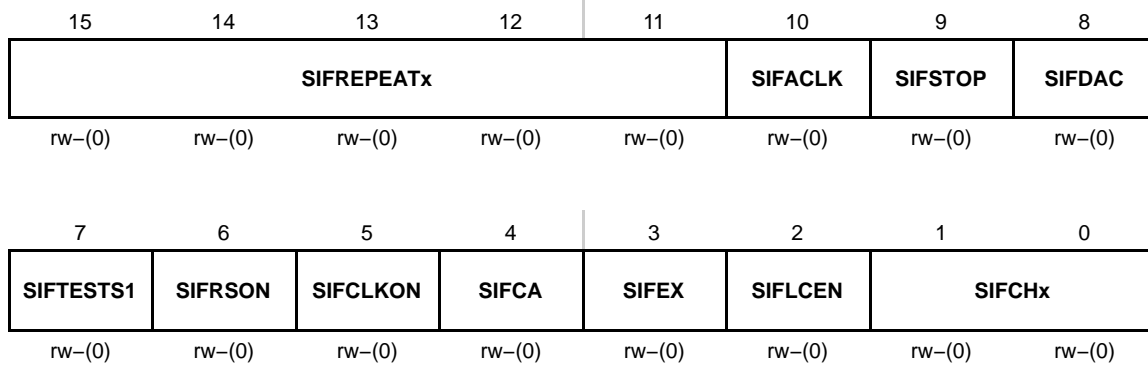
**SIFCTL5, Scan IF Control Register 5**

<b>SIFCNT3x</b>	Bits 15-8	Internal oscillator counter. SIFCNT3 counts internal oscillator clock cycles during one ACLK period when SIFFNOM = 0 or during four ACLK periods when SIFFNOM = 1 after SIFCLKGON and SIFCLKEN are both set
<b>SIFTSMRP</b>	Bit 7	TSM repeat mode 0 Each TSM sequence is triggered by the ACLK divider controlled with the SIFDIV3Ax and SIFDIV3Bx bits. 1 Each TSM sequence is immediately started at the end of the previous sequence.
<b>SIFCLKFQx</b>	Bits 6-3	Internal oscillator frequency adjust. These bits are used to adjust the internal oscillator frequency. Each increase or decrease of the SIFCLKFQx bits increases or decreases the internal oscillator frequency by approximately 5%. 0000 Minimum frequency : 1000 Nominal frequency : 1111 Maximum frequency
<b>SIFFNOM</b>	Bit 2	Internal oscillator nominal frequency 0 4 MHz 1 1 MHz
<b>SIFCLKGON</b>	Bit 1	Internal oscillator control. When SIFCLKGON = 1 and SIFCLKEN = 1, the internal oscillator calibration is started. SIFCLKGON is not used when SIFCLKEN = 0. 0 No internal oscillator calibration is started. 1 The internal oscillator calibration is started when SIFCLKEN = 1.
<b>SIFCLKEN</b>	Bit 0	Internal oscillator enable. This bit selects the high frequency clock source for the TSM. 0 TSM high frequency clock source is SMCLK. 1 TSM high frequency clock source is the Scan IF internal oscillator.

## SIFDACRx, Digital-To-Analog Converter Registers



<b>Unused</b>	Bits 15-10	Unused. These bits are always read as zero, and when written, do not affect the DAC output.
<b>DAC Data</b>	Bits 9-0	10-bit DAC data

**SIFTSMx, Scan IF Timing State Machine Registers**

<b>SIF REPEATx</b>	Bits 15-11	These bits together with the SIFACLK bit configure the duration of this state. SIFREPEATx selects the number of clock cycles for this state. The number of clock cycles = SIFREPEATx + 1.
<b>SIFACLK</b>	Bit 10	This bit selects the clock source for the TSM. 0 The TSM clock source is the high frequency source selected by the SIFCLKEN bit. 1 The TSM clock source is ACLK
<b>SIFSTOP</b>	Bit 9	This bit indicates the end of the TSM sequence. The duration of this state is always one high-frequency clock period, regardless of the SIFACLK and SIFREPEATx settings. 0 TSM sequence continues with next state 1 End of TSM sequence
<b>SIFDAC</b>	Bit 8	TSM DAC on. This bit turns the DAC on during this state when SIFDACON = 0. 0 DAC off during this state. 1 DAC on during this state.
<b>SIFTESTS1</b>	Bit 7	TSM test cycle control. This bit selects for this state which channel-control bits and which DAC registers are used for a test cycle. 0 The SIFTCH0x bits select the channel and SIFDACR6 is used for the DAC 1 The SIFTCH1x bits select the channel and SIFDACR7 is used for the DAC
<b>SIFRSON</b>	Bit 6	Internal output latches enabled. This bit enables the internal latches of the AFE output stage. 0 Output latches disabled 1 Output latches enabled

<b>SIFCLKON</b>	Bit 5	High-frequency clock on. Setting this bit turns the high-frequency clock source on for this state when SIFACLK = 1, even though the high frequency clock is not used for the TSM. When the high-frequency clock is sourced from the DCO, the DCO is forced on for this state, regardless of the MSP430 low-power mode. 0 High-frequency clock is off for this state when SIFACLK = 1 1 High-frequency clock is on for this state when SIFACLK = 1
<b>SIFCA</b>	Bit 4	TSM comparator on. Setting this bit turns the comparator on for this state when SIFCAON = 0. 0 Comparator off during this state 1 Comparator on during this state
<b>SIFEX</b>	Bit 3	Excitation and sample-and-hold. This bit, together with the SIFSH and SIFTEN bits, enables the excitation transistor or samples the input voltage during this state. SIFLCEN must be set to 1 when SIFEX = 1. 0 Excitation transistor disabled when SIFSH = 0 and SIFTEN = 1. Sampling disabled when SIFSH = 1 and SIFTEN = 0. 1 Excitation transistor enabled when SIFSH = 0 and SIFTEN = 1. Sampling enabled when SIFSH = 1 and SIFTEN = 0.
<b>SIFLCEN</b>	Bit 2	LC enable. Setting this bit turns the damping transistor off, enabling the LC oscillations during this state when SIFTEN = 1. 0 All SIFCHx channels are internally damped. No LC oscillations. 1 The selected SIFCHx channel is not internally damped. The LC oscillates.
<b>SIFCHx</b>	Bits 1-0	Input channel select. These bits select the input channel to be measured or excited during this state. 00 SIFCH0 01 SIFCH1 10 SIFCH2 11 SIFCH3

### Processing State Machine Table Entry (MSP430 Memory Location)

7	6	5	4	3	2	1	0
<b>Q7</b>	<b>Q6</b>	<b>Q5</b>	<b>Q4</b>	<b>Q3</b>	<b>Q2</b>	<b>Q1</b>	<b>Q0</b>

<b>Q7</b>	Bit 7	When Q7 = 1, SIFIFG6 will be set. When SIFQ6EN = 1 and SIFQ7EN = 1 and Q7 = 1, the PSM proceeds to the next state immediately, regardless of the SIFSTOP(tsm) signal and Q7 is used in the next-state calculation.
<b>Q6</b>	Bit 6	When Q6 = 1, SIFIFG5 will be set. When SIFQ6EN = 1, Q6 will be used in the next-state calculation.
<b>Q5</b>	Bit 5	Bit 5 of the next state.
<b>Q4</b>	Bit 4	Bit 4 of the next state.
<b>Q3</b>	Bit 3	Bit 3 of the next state.
<b>Q2</b>	Bit 2	When Q2 = 1, SIFCNT1 decrements if SIFCNT1ENM = 1 and SIFCNT2 decrements if SIFCNT2EN = 1.
<b>Q1</b>	Bit 1	When Q1 = 1, SIFCNT1 increments if SIFCNT1ENP = 1.
<b>Q0</b>	Bit 0	Bit 2 of the next state.